ARMY RESEARCH LABORATORY

Sensor Performance Evaluator for Battlefield Environments (SPEBE) C++ Application Programming Interface (API) Version 1.0

by David Marlin and Shane Thomas

ARL-TR-4363 **January 2008**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Army Research Laboratory
White Sands Missile Range, NM 88002- 5501

ARL-TR-4363 **January 2008**

Sensor Performance Evaluator for Battlefield Environments (SPEBE) C++ Application Programming Interface (API) Version 1.0

David Marlin
Computational and Information Sciences Directorate, ARL

and

Shane Thomas
Physical Science Laboratory, New Mexico State University

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2008	Final	

4. TITLE AND SUBTITLE
Sensor Performance Evaluator for Battlefield Environments (SPEBE) C++ Application Programming Interface (API) Version 1.0

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
David Marlin (U.S. Army Research Laboratory) and Shane Thomas (Physical Science Laboratory, NMSU)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Computational and Information Sciences Directorate
Battlefield Environment Division (ATTN: AMSRD-ARL-CI-ES)
White Sands Missile Range, NM 88002-5501

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-4363

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this report, a C++ library based on a compiled version of the Matlab-based Sensor Performance Evaluator for Battlefield Environments (SPEBE) is described. The library encapsulates the details of the Matlab infrastructure, including m-files and mxArray manipulation functions, so that the programmer can concentrate on the use of SPEBE rather than the details of compiled Matlab code. The high-level architecture of SPEBE is duplicated in the C++ class hierarchy, providing the programmer with classes that represent the functional grouping of data and computations found in SPEBE. These high-level classes are derived from low-level classes, which encapsulate the mxArrays and invoke the compiled m-files. Thus, the high-level classes are insensitive to changes in the compiled Matlab code resulting from Matlab revisions, while the low-level classes provide basic compiled Matlab functionality without regard to the overall SPEBE architecture. This greatly simplifies maintenance of the library in response to changes in either Matlab or SPEBE.

15. SUBJECT TERMS
Acoustic sensors, acoustic propagation, acoustic performance modeling, acoustic decision aid

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	SAR	104	David Marlin
U	U	U			19b. TELEPHONE NUMBER (*Include area code*) (575) 678-1524

Contents

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

Summary

The Sensor Performance Evaluator for Battlefield Environments (SPEBE) is a highly evolved decision aid and acoustic system developer tool to assist in performance modeling of acoustic sources and sensors on the battlefield as well as in non-battlefield and civil situations. It integrates acoustic and seismic propagation, atmospheric conditions, terrain, ground cover, geological effects, background noise, and extensible source and sensor modeling, under a point-and-click Graphical User Interface (GUI), to enable the user to predict a broad range of acoustic and seismic sensor performance characteristics. SPEBE incorporates a range of acoustic propagation capabilities including low-fidelity models for near-real-time battlefield calculations and high-fidelity models for advanced planning and sensor development. Users can add their own source and sensor models, as well as custom detection, estimation, and beam-forming algorithms.

In many cases, however, the computational capabilities of SPEBE are needed for existing, proposed, or under-development systems that include their own graphical environment, precluding the direct incorporation of the SPEBE GUI. Thus, what is needed is a callable library of SPEBE calculations distinct from the GUI. This library should closely duplicate the logic and execution of SPEBE, while accepting input parameters and providing output data via function calls rather than pop-up menus and displays.

That is the purpose of the SPEBE C++ Application Programming Interface (API). In addition to meeting the requirement to perform SPEBE calculations with the same logic and execution as SPEBE, the callable library should be maintainable, in the sense that changes in the GUI version of SPEBE should be easily incorporated into the API. Furthermore, the C++ classes should be extensible, in the sense that the user can readily adapt and extend them for smooth integration into another system. Finally, it should be independent of the underlying environment with which SPEBE was developed, Matlab. Users of the SPEBE API should not be burdened with aspects of compiled Matlab code, but rather should see only classes that encapsulate the data and computational aspects of SPEBE functionality.

This is accomplished by a two-level class hierarchy. The lower-level hierarchy encapsulates the data and methods of compiled Matlab code, providing manipulation of the data and execution of the functions without detailed knowledge of the Matlab. The upper-level hierarchy, derived from the lower-level, encapsulates the logical data and computation architecture of SPEBE. In this way, the lower-level classes can be modified to accommodate changes in Matlab and the Matlab compiler without impacting the SPEBE API visible to the user. Similarly, changes in the API can be made without regard to the details of compiled Matlab code.

INTENTIONALLY LEFT BLANK.

1. Introduction

This report details an Application Programming Interface (API) that provides a developer with the ability to integrate Sensor Performance Evaluator for Battlefield Environments (SPEBE) (*1–11*) computations into systems which for one reason or another can't use the SPEBE Graphical User Interface (GUI). It can also be used as the foundation for a specialized GUI built in Java, Microsoft Foundation Class (MFC), or another such tool. It does this without forcing the developer to master the use of compiled Matlab* code by performing the following:

- It encapsulating the basic Matlab data structure, the mxArray.
- It wraps the compiled m-files.

The overall structure of the API is true to the architecture of SPEBE and is based on several Matlab structures, which group the SPEBE data (section 2.1). These structures are encapsulated by the API classes, and appropriate m-files are included in the class methods in order to set and retrieve data as necessary. *All calculations are performed by SPEBE m-files*. Thus, the API simply "wraps" the SPEBE structures, providing the programmer access to certain parameters, and passes these structures back and forth between the SPEBE m-files for calculations. This ensures that the API performs *exactly* the same calculations as the GUI version of SPEBE.

The API can be divided into two parts:

- a low-level class hierarchy that manages the compiled Matlab machinery, and
- a high-level set of derived classes that present the SPEBE interface to the end-user without regard to the specific details of compiled Matlab code.

The heart of the API is the **CMXArray**, which encapsulates a Matlab mxArray; however, the **CMXArray** class is never used directly by the developer. Instead, there are a number of classes, such as the following:

- derived classes (such as **CMXNumeric**, and **CMXStructure**, etc.), which encapsulate particular types of mxArray, and
- further derived classes (such as **CMetInfo**, **CDomain**, etc.), which in turn encapsulate specific SPEBE data structures.

The general naming convention is as follows:

*Matlab is a registered trademark of The MathWorks, Inc.

- A lower-level derived class that directly encapsulates a particular mxArray type (such as **CMXNumeric**, and **CMXStructure**, etc.) is prefixed with “CMX”.
- A higher-level derived class that encapsulates one or more specific SPEBE structures (such as **CMetInfo**, **CDomain**, etc.) is prefixed simply with “C”.
- A higher-level derived class that does not directly encapsulate any SPEBE structure (such as **PerformanceCalculator**) has no specific prefix.

In general, the lower-level, CMX classes incorporate all of the mxArray functionality, insulating the higher-level classes from any specific compiled Matlab dependency. Thus, the higher-level, SPEBE-specific classes should look the same regardless of changes in the Matlab compiler operation and behavior of mxArrays and the associated Matlab mx function calls.

Furthermore, by encapsulating the SPEBE mxArray structures in C++ classes rather than defining parallel structures in C, the internal coding of the API is made less sensitive to revisions in the SPEBE structures. As long as the *.mat* files are current, the encapsulated mxArrays will contain the proper fields, without the need to modify any C structures. If a new field requires modification, then an appropriate method will have to be added to the associated higher-level class, but the methods in **CMXStructure** make this extremely simple.

Thus the principal advantages of the C++ API are as follows:

- The low-level interface is insensitive to changes in SPEBE.
- The high-level interface is independent of changes in Matlab and the Compiler Toolbox.
- The high-level interface is relatively easily modified to incorporate changes in SPEBE.

The classes of the API are depicted and described in more detail in the subsequent sections and also in the specific class documentation.

NOTE: Portions of the documentation that are specific to the lower-level interface or details of the Matlab SPEBE implementation are generally preceded by “Internal behavioral details” and can be ignored by the user who is only interested in the high-level SPEBE functionality. These sections are intended primarily for developers of the API, as opposed to end users. However, some insight can be gained into the underlying SPEBE architecture and the logic behind the API organization, so these sections can be beneficial to the end user as well.

2. Basic Architecture

2.1 SPEBE Structures

The basic SPEBE structures are as follows:

- Location and computational grid descriptors:
  - Origin, Lx, Ly: geographical location and extent of the computational domain
  - Nx, Ny: dimensions of the computational grid
- Environment descriptors:
  - MetInfo: meteorological parameters and profiles
  - GroundInfo: ground impedance characterization
  - TerrainInfo
  - BarrierInfo
- Source and sensor descriptors:
  - SourceInfo
  - RcvrInfo
- Propagation calculation descriptors:
  - PropInfo: acoustic and seismic propagation models, associated parameters
  - AcsPropTab, SeisPropTab: table parameters associated with some of the propagation models

There are hundreds of SPEBE m-files, but all are basically of two types:

- GUI and graphics
- Acoustic and meteorological calculations

The API separates the GUI and graphics from the calculations and makes the latter available to the programmer. In the discussion that follows, reference will sometimes be made to the GUI, but this is only for illustrative purposes.

When the GUI version of SPEBE starts up, most of the SPEBE structures are created with default values for many, but not all, of their fields. Also, the GUI can be configured so that most of the structures will have values read from a set of *.mat* files. The associated C++ API classes mimic this behavior, by setting the same default parameters upon instantiation, and also provide constructors that will read values from specified *.mat* files.

The parameters associated with a specific structure can be modified within Matlab by editing the associated *.mat* files, but this is not the usual mode of operation. Instead, in the GUI, the user selects options from pull-down menus. Internally, m-files are then called that set the appropriate fields of the appropriate structures. In the C++ API, these fields are set by methods (functions) of the associated classes. Sections 2.2 through 2.4 describe this in more detail.

2.2 Location, Environment, and Computational Grid

The **CDomain**, **CMetInfo**, **CGroundInfo**, **CTerrainInfo**, and **CBarrierInfo** classes encapsulate the SPEBE structures involved with the location, environment, and computational grid setup. Details of each can be found under their class descriptions. **CDomain** incorporates the location and computational grid parameters, while the other classes are self-explanatory. The following example of the **CMetInfo** class illustrates the general relationship between a SPEBE structure, the related m-files invoked by the GUI, and the associated C++ class.

To specify the predefined meteorological case “Mostly Sunny, Light Wind,” the GUI user would navigate the pull-down menus by selecting Weather >> Pre-Defined Meteorological Cases >> Mostly Sunny, Light Wind.

This would result in a menu-callback that performs the following:

1. Prompts the user for these parameters:
   - wind direction
   - surface temperature and the height of the temperature measurement
   - surface humidity and the height of the humidity measurement
2. Sets several parameters in the MetInfo structure.
3. Calls an m-file function to generate similarity profiles using the MetInfo parameters as well as certain GroundInfo parameters.

The C++ API programmer would instead call the following:

> **CMetInfo::SetPredefinedMeteorology(PredefinedMet metCase, double dWindDir, double dTemp, double dTempHeight, double dRelHumid, double dRelHumHeight, CGroundInfo *ground)**

With the argument *metCase* selected from the enumerated type:

```
PredefinedMet {SunnyLightWind=2, SunnyModerateWind, SunnyStrongWind,
               NightLightWind, NightModerateWind, NightStrongWind,
               OvercastLightWind, OvercastModerateWind,
               OvercastStrongWind};
```

This method then performs the following:

1. Sets the appropriate fields within the encapsulated MetInfo structure.
2. Calls the same (compiled) m-file function to generate the same similarity profiles using the encapsulated MetInfo parameters as well as the encapsulated GroundInfo parameters from the *ground* argument.

2.3 Propagation Calculations

Once all the appropriate parameters are set, including the meteorology (**CMetInfo**), seismology (**CSeismicInfo**), and ground characterization (**CGroundInfo**), any of the propagation models can be invoked to determine transmission losses. These are required for sensor calculations and may also be of interest in their own right.

Within the GUI, a user can invoke transmission loss calculation *explicitly*, by requesting transmission loss, or *implicitly*, by requesting a sensor performance calculation such as detection probability. In the latter case, SPEBE will generate acoustic and/or seismic *propagation tables* and save them to files. These tables contain transmission loss data over the computational grid at a set of frequencies specified in the AcsPropTab and SeisPropTab structures. They are saved in *temporary* files, whose names are specified within the AcsPropTab and SeisPropTab structures.

The user also has the option to explicitly save the tables as *permanent* files, and the permanent file names will be stored in the AcsPropTab and SeisPropTab structures. The only difference between the temporary files and the permanent files is that the former are overwritten whenever a new transmission loss calculation is made, while the latter can be reused any time.

The API has two restrictions on the GUI behavior described above:

1. The propagation calculations must be *explicitly* requested.
2. The resulting propagation tables must be stored in a *permanent* file.

For example, to invoke the acoustic propagation calculations and save the acoustic propagation table, the programmer would make the following calls:

CPropagator::SetAcsPropMod(AcsPropModel acsPropModel);
CPropagator::SetFilePrefix(char *filePrefix, char *fileDir);
CPropagator::GenerateAcousticPropagationTable(double sourceHeight, double receiverHeight, CMetInfo *metInfo, CDomain *domain);

NOTE: The *filePrefix* is not the full file name and should not include a file type; it will be appended with "_acs.mat" for an acoustic propagation table and "_seis.mat" for a seismic table.

2.4 Sensor Calculations

For a detailed description of SPEBE sensor calculations, see the description of the **PerformanceCalculator** class (section 4.18).

Internal behavioral details.

NOTE: With regard to the distinction between GUI m-files and calculation m-files, there is some overlap when graphics commands, such as waitbars, are embedded in the calculations. More importantly, in SPEBE version 1.2 there were some instances where calculations, and the setting of parameters, were embedded in the GUI-related functions—particularly in some of the callbacks and menu-prompting routines. This makes it difficult to insure the API behaves exactly as the GUI version since it requires the embedded calculations and parameter-setting to be duplicated in separate code within the API. Any subsequent changes to these m-files would the have to be reflected in the API code. This invites mistakes and violates one of the basic principles of the API: *All calculations are performed by SPEBE m-files*. Thus, a few changes were made to the SPEBE m-files in order to move the offending calculations and parameter-setting into calculation-specific m-files.

2.5 Some Examples

Every process must begin with the call:

```
abfamlbmInitialize();
```

Prior to making any SPEBE API calls, it should end with the following:

```
abfamlbmTerminate();
```

The following code segment provides some examples of the use of the API:

```
// initialize the compiled Matlab dll
abfamlbmInitialize();

//instantiate a new CDomain object from the file "Terrain.mat",
// and set the location, grid, and terrain elevations
CDomain *dInfo = new CDomain("Terrain.mat");
double lat = 33.0265;
double long = -106.139;
double lx = 1000;
double ly = 1000;
int timeZone = 6;
dInfo->SetLocation(lat, long ,lx ,ly , timeZone);
int nx = 100;
```

```
int ny = 100;
dInfo->SetGridDimensions(nx, ny);
dInfo->LoadDTEDHeights("f:\\");

// instantiate a new default CMetInfo object
// and select a predefined meteorological case,
// using the CGroundInfo object contained within the CDomainInfo
object above
mInfo = new CMetInfo();
double windDir = 180;
double surfTemp = 25;
double tempHt =1;
double relHum = .8;
double humHt = 1;
mInfo->SetPredefinedMeteorology(SunnyLightWind, windDir, surfTemp,
                        tempHt, relHum, humHt,
                        dInfo->GetGroundInfo());

// instantiate a new CSourceInfo object, containing only one source,
// from the file "HMMWV, 0 mph.mat",
CSourceInfo *srInfo = new CSourceInfo("HMMWV, 0 mph.mat");

// instantiate a new CRcvrInfo object, containing only one receiver,
// from the file "ISO Standard (good hearing).mat"
 CRcvrInfo* humanInfo = new CRcvrInfo("ISO Standard (good
                        hearing).mat");

// instantiate a new CRcvrInfo object, containing only one receiver,
// from the file "GeoSpace 1-Hz geophone.mat"
CRcvrInfo* geophoneInfo = new CRcvrInfo("GeoSpace 1-Hz geophone.mat");

// instantiate a new CPropagator from the file "Prop.mat",
// set the propagation models and file names,
// and generate propagation tables for the source and sensor
// heights of the CSourceInfo and CRcvrInfo objects instantiated above
pInfo = new CPropagator("Prop.mat");
pInfo->SetFilePrefix("test_table", "D:\\SPEBE\\SPEBEruntime\\");
pInfo->SetAcsPropMod(SCAPE);
pInfo->SetSeisPropMod(HARVEY_MODE_SUM);
pInfo->GenerateAcousticPropagationTable(srInfo->GetHeight(0),
                        humanInfo->GetHeight(0), mInfo, dInfo);
pInfo->GenerateSeismicPropagationTable(srInfo->GetHeight(0),
                        geophoneInfo->GetHeight(0), dInfo);

// instantiate a new PerformanceCalculator, in ReceiverCentric mode,
```

```
// for the human receiver and HMMWV above,
// and generate detection probabilities.
// Note that the CRcvrInfo object contains only one receiver, with
index 0.
// This is the receiver activated in the call Activate(0) below.
// Also, the CSourceInfo object contains only on source, with index 0.
// This is the source selected in the call GetPD(0) below.
PerformanceCalculator* theCalculator = new
                PerformanceCalculator(humanInfo, srInfo,
                mInfo, dInfo, pInfo, RcvrCentric);
theCalculator->Activate(0);
ResultGrid *theHumanGrid = theCalculator->GetPD(0);
delete theCalculator;

// instantiate a new PerformanceCalculator, in ReceiverCentric mode,
// for the geophone receiver and HMMWV above,
// and generate detection probabilities
theCalculator = new PerformanceCalculator(geophoneInfo, srInfo, mInfo,
                        dInfo, pInfo, RcvrCentric);
theCalculator->Activate(0);
ResultGrid* theGeophoneGrid = theCalculator->GetPD(0);

// clean house
abfamlbmTerminate();
```

3. Class List

Table 1 lists the classes, structures, unions, and interfaces with brief descriptions.

Table 1. Class list.

Name	Description
CBarrierInfo	Encapsulates the SPEBE BarrierInfo structure.
CDomain	Encapsulates the SPEBE Origin, Nx, Ny, Lx, Ly, heights, xdom, and ydom structures. Also contains **CGroundInfo**, **CTerrainInfo**, **CSeismicInfo,** and **CBarrierInfo** objects for convenience. All these can be initialized from a single call to LoadFile, using a *Terrain.mat* file.
CGroundInfo	Encapsulates the SPEBE GroundInfo structure, which contains a number of parameters characterizing the ground with regard to its acoustic impedance, effect on wind, and thermal absorption and radiation characteristics. However, these parameters are not manipulated within the scope of the SPEBE API, but instead are specified via a collection of files describing various common ground types, including • Urban, • Suburban, • Asphalt, • Gravel, • Sand, • Brush, • Forest, • Short grass, • Long grass, • Open water, • Ice, and • Snow.
CMetInfo	Encapsulates the SPEBE MetInfo structure.
CMXArray	Encapsulates a Matlab mxArray, and also includes various utilities for manipulating mxArrays.
CMXMultiStructure	Adds methods to manage arrays of identical structures. It is intended to support Source and Rcvr arrays.
CMXNumeric	Encapsulates numeric type mxArrays, providing various methods to manipulate the data.
CMXSingleStructure	Streamlines **CMXStructure** for use with single-element Matlab structures. Arrays of Matlab structures are managed by **CMXMultiStructure**.
CMXString	Encapsulates string mxArrays.
CMXStructure	Encapsulates a Matlab array of structures.

Table 1. Class list (continued).

Name	Description
CNoizInfo	Encapsulates the SPEBE NoizInfo structure, which contains three fields: the acoustic spectrum, the seismic spectrum, and a flag to indicate whether or not to include acoustic wind noise. The acoustic and seismic spectra are in turn described by structures defining the spectral bands in terms of lower and upper frequency, slope, and loudness.
CPropagator	Encapsulates the SPEBE PropInfo structure. Also manages acoustic and seismic PropTab structures internally, and includes a **CNoizInfo** object for convenience. All these can be initialized with from a single call to LoadFile, from a *Prop.mat* file.
CRcvrInfo	Encapsulates the SPEBE RcvrInfo structure.
CSeismicInfo	Encapsulates the SPEBE SeismicInfo structure, which defines vertical profiles of P and S-wave velocity and attenuation as well as density.
CSourceInfo	Encapsulates the SPEBE SourceInfo structure.
CTerrainInfo	Encapsulates the SPEBE TerrainInfo structure, which contains parameters characterizing random terrain elevations. Random terrain is not currently supported by the API, so this class has no methods for specifying, manipulating, or retrieving parameters. It is provided because it is needed internally by some of the terrain and domain related m-files which are part of the compiled library.
MFileHandler	This Helper class manages input and output mxArray arrays for calling m files.
PerformanceCalculator	Performs the sensor performance calculations of SPEBE.
ResultGrid	Encapsulates two-dimensional (2-D) gridded data returned by SPEBE calculations, such as detection probability. Also generates contour plots of the encapsulated data.

4. Class Hierarchy

4.1 CMXArray Class Reference

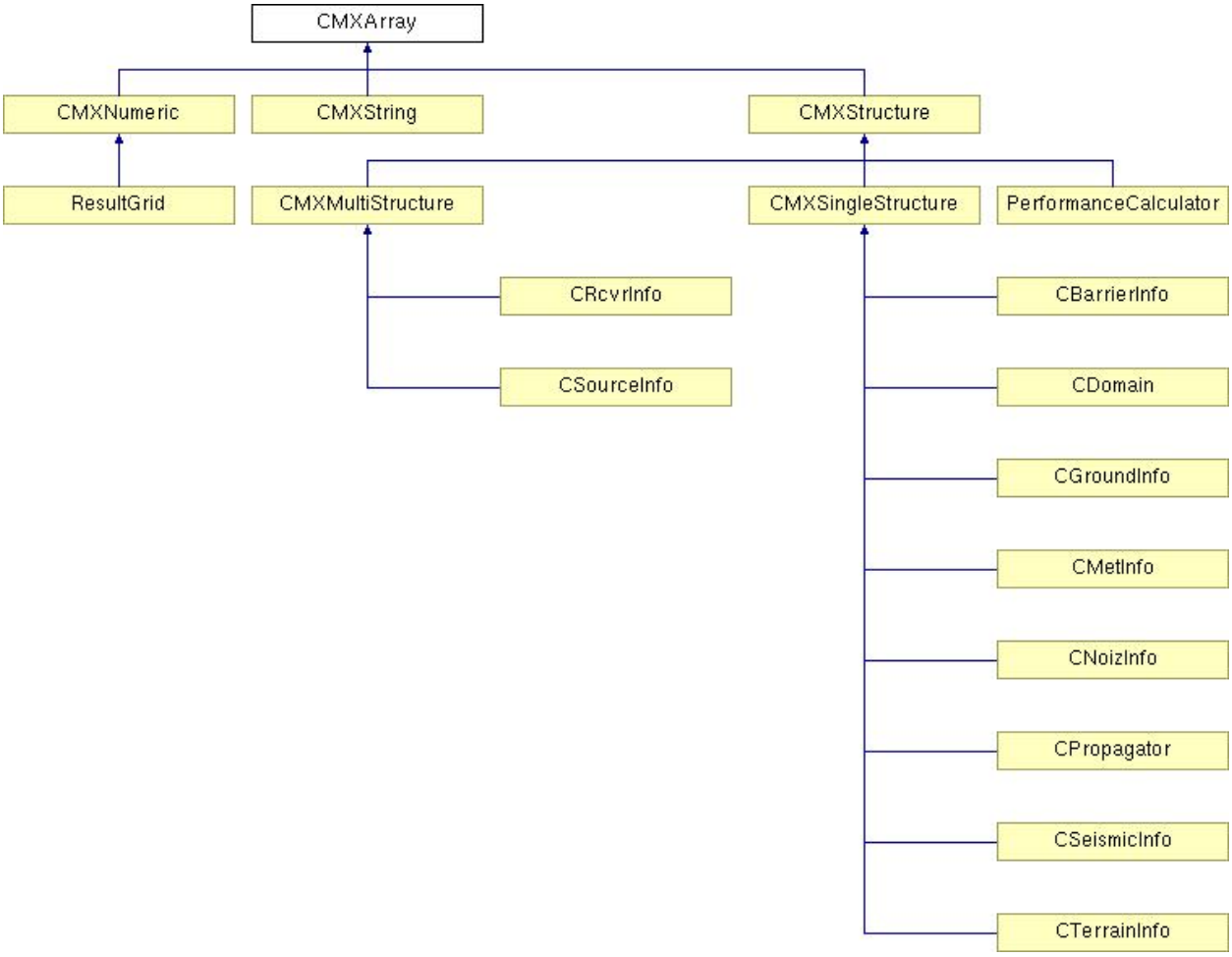

Figure 1. Inheritance diagram for **CMXArray**.

The **CMXArray** class encapsulates a Matlab mxArray and also includes various utilities for manipulating mxArrays.

The encapsulated array can be any Matlab array type. This class contains various utilities for working with mxArrays in general, such as a method to load an mxArray from a file, methods to encapsulate mxDuplicate and mxDestroy, and a class to manage arguments for compiled m-files.

USE CASES: This class has no public constructors. It is intended as the base class for subclasses, which will encapsulate different types of mxArrays:

- **CMXNumeric**
- **CMXStructure**

Warning: In all cases, the internal mxArray should be encapsulated. The two methods, **::SetArray** and **::GetArray**, will deep copy the input/output array for this purpose. The **::SetArray** method will also do an mxDestroy on the existing encapsulated mxArray, if it exists. These methods should be used AT ALL TIMES, with the following exceptions:

- Direct subclasses (**CMXNumeric** and **CMXStructure**) can access the encapsulated mxArray directly to obtain information such as the field names, the number of fields, number of elements, type of mxArray, etc.
- Direct subclasses can assign the encapsulated mxArray directly in the constructors, and delete it in the destructors.
- **CMXStructure** can access the encapsulated mxArray directly to modify fields within a structure array.

Second-generation derived classes (i.e., subclasses of **CMXNumeric** and **CMXStructure**) should refrain from accessing the encapsulated mxArray directly and should also refrain from creating or deleting any mxArrays directly.

Failure to adhere to these practices may result in poor memory use and crashes.

4.1.1 Constructor and Destructor Documentation

```
CMXArray          ( mxArray *   theArray,
                    const char * arrayName
                  )
```

This function encapsulates a deep copy of theArray as the internal mxArray, m_mxArray, and gives it the specified name, arrayName.

Parameters:

- *theArray* – the mxArray to be copied
- *arrayName* – the array name; it will be copied into the m_arrayName attribute

~CMXArray(void) [virtual]

This function invokes a Matlab command to completely free all memory allocated to the encapsulated mxArray. NOTE: Always delete CMXArrays.

4.1.2 Member Function Documentation

const char * GetArrayName(void)

This function returns a pointer to the array name, arrayName.

Returns: A const char* pointer to the **CMXArray** array name. This doesn't call malloc (since the return is const), so do not call free.

int GetNumElements(void)

This function returns the number of elements in the encapsulated mxArray.

Returns: The number of elements

void SaveToFile(const char * *fileName*) [virtual]

This function saves the encapsulated mxArray to a *.mat* file.

Warning: An existing file will be overwritten without warning.

Parameters:

- *fileName* – the file in which to save the encapsulated data

This function is reimplemented in **CDomain** and **CPropagator**.

4.2 CMXNumeric Class Reference

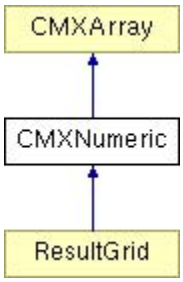

Figure 2. Inheritance diagram for **CMXNumeric**.

The **CMXNumeric** class encapsulates numeric type mxArrays, providing various methods to manipulate the data.

4.2.1 Constructor and Destructor Documentation

CMXNumeric(CMXNumeric * *theArray*)

This function is a copy constructor that makes a deep copy of the encapsulated parameter array.

Parameters:

- *theArray* – the **CMXNumeric** object to copy

CMXNumeric(CMXArray * *theArray*)

This function is another copy constructor, intended primarily for returns from GetField.

Parameters:

- *theArray* – the **CMXArray** object to copy

CMXNumeric (const char * *fileName*,
const char * *arrayName*
)

This function creates a new numeric array from the specified file, with the given name.

Warning: This returns an empty array if no data found or if data are not strictly numeric.

Parameters:

- *fileName* – the file containing the numeric data
- *arrayName* – the name to give the new object

CMXNumeric (double * *data*,
int *size*
)

This function creates a numeric array from the specified double array.

Parameters:

- *data* – points to the array holding the data to be copied into the mxArray
- *size* – the number of elements in the double array

```
CMXNumeric( double *      data,
            int           size,
            const char *  arrayName
          )
```

This function creates a numeric array from the specified double array, with the specified name.

Parameters:

- *data* – points to the array holding the data to be copied into the mxArray
- *size* – the number of elements in the double array
- *arrayName* – the name to give the new object

```
CMXNumeric( double data )
```

This function creates a numeric array from the specified scalar double.

Parameters:

- *data* – the value to be put into the mxArray

```
CMXNumeric  ( double      data,
              const char * arrayName
            )
```

This function creates a numeric array from the specified scalar double.

Parameters:

- *data* – the value to be put into the mxArray
- *arrayName* – the name to give the new object

```
CMXNumeric( int data )
```

This function creates a numeric array from the specified scalar integer.

Parameters:

- *data* – the value to be put into the mxArray

CMXNumeric (int *data,*
const char * *arrayName*
)

This function creates a numeric array from the specified scalar integer.

Parameters:

- *data* – the value to be put into the mxArray
- *arrayName* – the name to give the new object

4.2.2 Member Function Documentation

double * GetData(void)

This function returns a copy of the internal array. Use the command **free** to release the memory when no longer needed.

Returns: A pointer to a copy of the internal array

double GetScalar(void)

This function returns the value of the internal scalar or the first element of the internal array

Returns: The value as a scalar double

bool IsEqual(CMXNumeric * *theArray*) `[virtual]`

This function compares with another **CMXNumeric** to see if their internal arrays contain the same values.

Parameters:

- *theArray* – the **CMXNumeric** array to be compared

Returns: True, if they are equal; false, if they are not

void Print(void)

This function prints to the screen the numeric value(s).

4.3 ResultGrid Class Reference

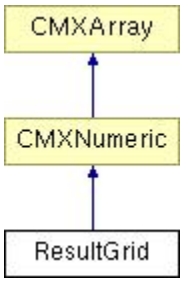

Figure 3. Inheritance diagram for **ResultGrid**.

The **ResultGrid** class encapsulates two-dimensional gridded data returned by SPEBE sensor performance calculations and generates contour plots of the data. It also keeps track of the computational grid coordinates, so that the data can be referenced to its location within the computational grid for contour generation.

See also **PerformanceCalculator** (section 4.18).

4.3.1 Constructor and Destructor Documentation

ResultGrid(ResultGrid * *theArray*)

This function is a copy constructor.

Parameters:

- *theArray* – the **ResultGrid** to be copied

```
ResultGrid    ( double *      theData,
                int           numRows,
                int           numCols,
                CDomain *     theDomain,
                const char *  arrayName
              )
```

This function instantiates a new **ResultGrid** with data contained in a linear double* array. NOTE: This assumes the data points are located on the computational grid defined in *theDomain*. If they are not, the results are unpredictable.

Parameters:

- *array* – the returned data to be incorporated into the **ResultGrid**
- *theDomain* – the **CDomain** object associated the data returned by the m-file
- *arrayName* – the name to be associated with the data array

4.3.2 Member Function Documentation

void GenerateContours(double *contourValue*)

This function generates a set of contours for the encapsulated data, for the specified *contourValue*.

Parameters:

- *contourValue* – the contour value

double * GetContourEasting(int *contour*)

This function returns the eastings of a specified contour generated by GenerateContours.

Parameters:

- *contour* – specifies the desired contour

Returns: An array of eastings for the points in the specified contour

double * GetContourLat(int *contour*)

This function returns the latitudes of a specified contour generated by GenerateContours.

Parameters:

- *contour* – specifies the desired contour

Returns: An array of latitudes for the points in the specified contour

int GetContourLength(int *contour*)

This function returns the length of a specified contour generated by ResultGrid::GenerateContours.

Parameters:

- *contour* – specifies the desired contour

Returns: The number of points in the specified contour

double * GetContourLong(int *contour*)

This function returns the longitudes of a specified contour generated by GenerateContours.

Parameters:

- *contour* – specifies the desired contour

Returns: An array of longitudes for the points in the specified contour

double * GetContourNorthing(int *contour*)

This function returns the northings of a specified contour generated by GenerateContours.

Parameters:

- *contour* – specifies the desired contour

Returns: An array of northings for the points in the specified contour

double GetGridMax(void)

This function returns maximum value within the grid.

Returns: The maximum grid value

double GetGridMin(void)

This function returns minimum value within the grid.

Returns: The minimum grid value

double GetGridXY (int *xx*,
int *yy*
)

This function returns the value of the specified grid point.

Parameters:

- *xx* – X index of specified grid point
- *yy* – Y index of specified grid point

Returns: The value of the grid point with the specified indices

int GetNumContours(void)

This function returns the number of contours generated by **ResultGrid::GenerateContours**.

Returns: The number of contours

int GetNumGridX(void)

This function returns the number of grid points in the X (east-west) dimension.

Returns: The number of points in the X dimension

int GetNumGridY(void)

This function returns the number of grid points in the Y (north-south) dimension.

Returns: The number of points in the Y dimension

4.4 CMXString Class Reference

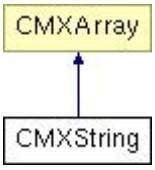

Figure 4. Inheritance diagram for **CMXString**.

The **CMXString** class encapsulates string mxArrays.

4.4.1 Constructor and Destructor Documentation

CMXString(const char * *data*)

This function creates a new string array containing a copy of the specified string.

Parameters:

- *data* – the string to be copied into the new array

4.4.2 Member Function Documentation

void Print(void)

This function prints the string to the standard output.

4.5 CMXStructure Class Reference

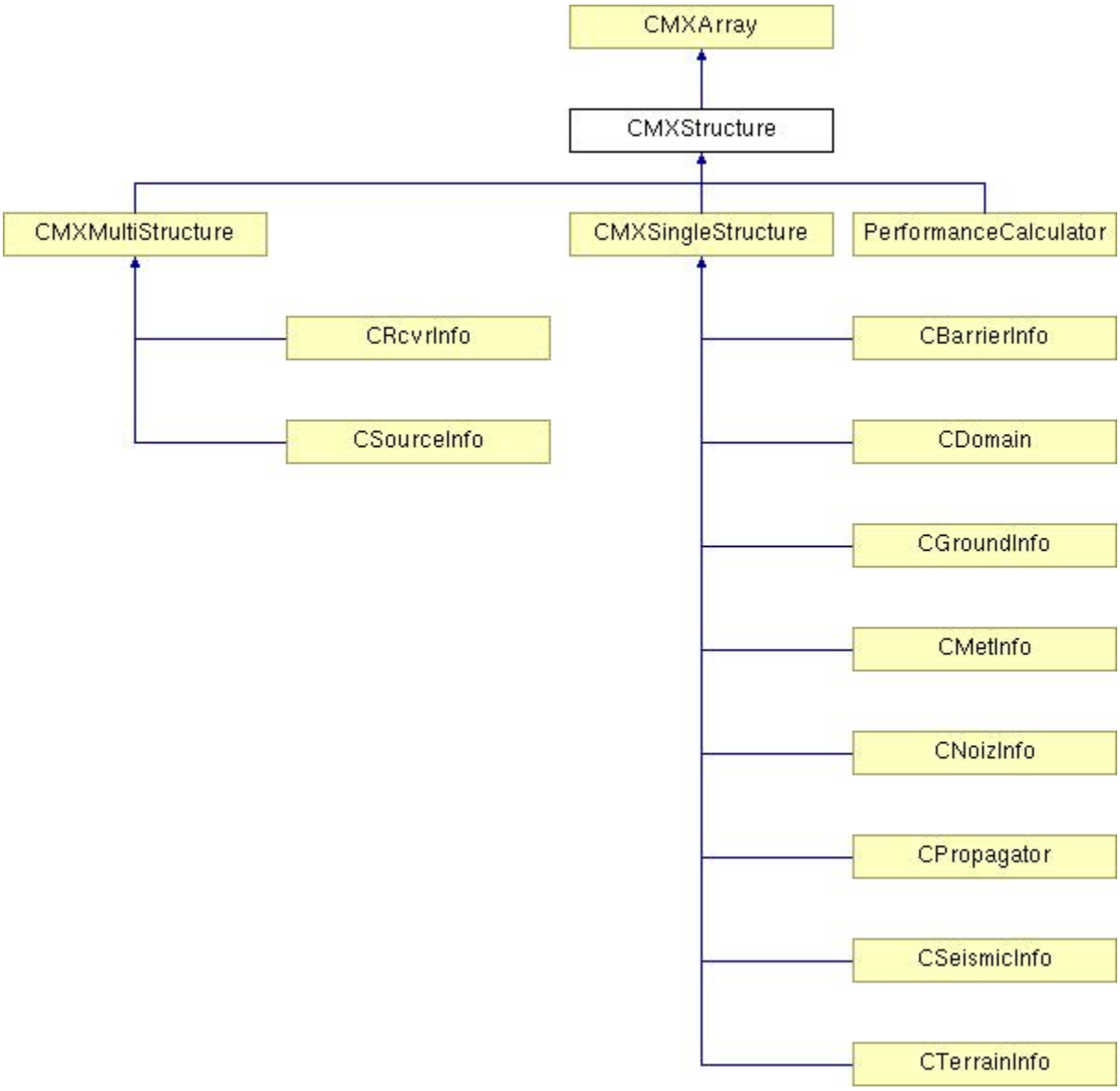

Figure 5. Inheritance diagram for **CMXStructure**.

The **CMXStructure** class encapsulates a Matlab array of structures.

The array can contain one or more structure elements. The structures in each element are composed of identical fields, but each element contains independent values for each field of the structure. It also includes utility functions for creating and maintaining matlab struct arrays.

USE CASES: This class has no public constructors. It is intended as the base class for three subclasses:

- **CMXMultiStructure** for the source and receiver structures, which can represent multiple elements

- **CMXSingleStructure** for most of the SPEBE data structures
- **PerformanceCalculator** to invoke sensor performance calculations

See also **CMXArray**, **CMXSingleStructure**, **CMXMultiStructure**, and **PerformanceCalculator**.

4.5.1 Member Function Documentation

**double GetDouble(int *arrayElement,*
const char * *fieldName*
)**

This function returns the specified double.

Parameters:

- *arrayElement* – integer identifying the index of the element to be retrieved
- *fieldName* – string identifying the name of the field to be set

Returns: The value requested

**CMXArray * GetField(int *arrayElement,*
const char * *fieldName*
)**

This function returns the value of a given field for a given element of the array.

Parameters:

- *arrayElement* – integer identifying the index of the desired element
- *fieldName* – string identifying the name of the desired field

Returns: The **CMXArray** pointer to the data

const char * GetFieldName(int *fieldIndex*)

This function returns the name of the specified field.

Parameters:

- *fieldIndex* – the index to the specified field

Returns: A constant string containing the fieldName

int GetNumFields(void)

This function returns the number of fields in the structure.

Returns: The number of fields

**const char * GetString(int *arrayElement,*
const char * *fieldName*
)**

This function returns a const char* of the specified field.

Warning: GetString uses a single string buffer to return a const char* string. Thus, the contents of the buffer will be overwritten upon the next call to GetString. If the returned string must persist after the next call, then copy it using strdup.

Parameters:

- *arrayElement* – integer identifying the index of the element to be retrieved
- *fieldName* – string identifying the name of the field to be retrieved

Returns: The const char* of the requested field

4.6 CMXMultiStructure Class Reference

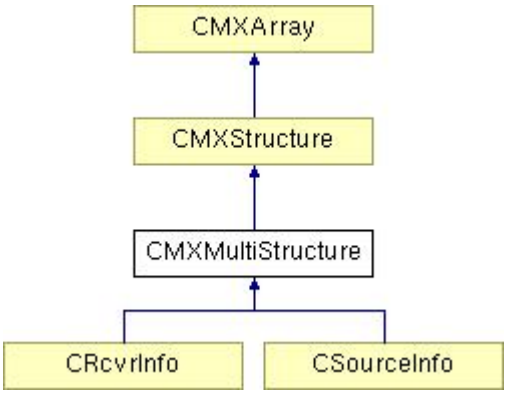

Figure 6. Inheritance diagram for **CMXMultiStructure**.

The **CMXMultiStructure** class adds methods to manage arrays of identical structures. It is intended to support Source and Rcvr arrays.

USE CASES:

- Load from file into a specified element.
- Set specific fields of all elements.
- Manually set coordinates of individual elements.

It also includes a constructor for a single-element, single-field structure containing only the field zcoord. This is for use with GenPropTab, which needs SourceInfo and RcvrInfo structures as arguments, but only uses the zcoord.

In any case, there are NO DEFAULTS for Source and Rcvr arrays, so all values must be provided in the files.

See also **CMXStructure**, **CSourceInfo**, and **CRcvrInfo**.

4.6.1 Member Function Documentation

double GetHeight(int *element*)

This function returns the height of the specified element.

Parameters:

- *element* – the index of the specified element

Returns: The height

double GetXCoord(int *element*)

This function returns the longitude, or X coordinate, of the specified element

Parameters:

- *element* – the index of the specified element

Returns: The longitude in signed decimal, positive for east

double GetYCoord(int *element*)

This function returns the latitude, or Y coordinate, of the specified element

Parameters:

- *element* – the index of the specified element

Returns: The latitude, in signed decimal, positive for north

```
void SetCoordinates ( int      element,
                      double lat,
                      double lon
                    )
```

This function sets the location of the specified element.

Parameters:

- *element* – the index of the specified element
- *lat* – the latitude of the specified element, in signed decimal, positive for north
- *long* – the longitude of the specified element, in signed decimal, positive for east

```
void SetHeight    ( int      element,
                    double height
                  )
```

This function sets the height of the specified element.

Parameters:

- *element* – the index of the specified element
- *height* – the height of the specified element

```
void SetHeight( double height )
```

This function sets the height of all elements in the array.

Parameters:

- *height* – height used for all elements of the array

4.7 CRcvrInfo Class Reference

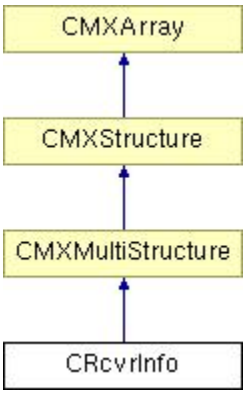

Figure 7. Inheritance diagram for **CRcvrInfo**.

The **CRcvrInfo** class encapsulates the SPEBE SourceInfo structure.

USE CASES:

- Inherited use cases from **CMXMultiStructure**
- Create structure by loading from a list of specified file names

Internal behavioral details.

For a detailed description of the encapsulated Matlab SourceInfo structure, see appendix I.

4.7.1 Constructor and Destructor Documentation

CRcvrInfo(double *height*)

This function creates a single receiver with only the height (Z coordinate) field.

Parameters:

- *height* – the height (Z coordinate) of the single element

CRcvrInfo(const char * *fileName*)

This function creates a single receiver from file.

Parameters:

- *fileName* – the name of the Matlab *.mat* ReceiverInfo file describing the receiver

CRcvrInfo(CRcvrInfo * *rcvrInfo*)

This function creates a single receiver from a deep copy of an existing single-element **CRcvrInfo**.

Parameters:

- *rcvrInfo* – the existing **CRcvrInfo** object

CRcvrInfo(int *numRcvrs,*
const char * *fileName*
)

This function creates multiple, identical receivers from a file.

Parameters:

- *numRcvrs* – the number of identical receivers
- *fileName* – the name of the Matlab *.mat* ReceiverInfo file describing the receiver

CRcvrInfo(int *numRcvrs,*
CRcvrInfo * *rcvrInfo,*
int *sourceElement*
)

This function creates multiple, identical receivers from a deep copy of a single element of an existing multiple-element **CRcvrInfo**.

Parameters:

- *numRcvrs* – the number of identical receivers to be created
- *rcvrInfo* – the existing **CRcvrInfo** object
- *sourceElement* – the index of the element of the existing **CRcvrInfo** object to be copied

4.7.2 Member Function Documentation

CRcvrInfo * GetRcvrInfo(int *elementNumber*)

This function returns a single-element **CRcvrInfo** from a deep copy of the specified element.

Parameters:

- *arrayElement* – the index of the element to be copied

**void LoadFile (int *arrayElement,*
const char * *fileName*
) [virtual]**

This function loads a single receiver element from file.

Parameters:

- *arrayElement* – the index of the element to be loaded from file
- *fileName* – the name of the Matlab *.mat* ReceiverInfo file describing the receiver

This function implements **CMXMultiStructure**.

4.8 CSourceInfo Class Reference

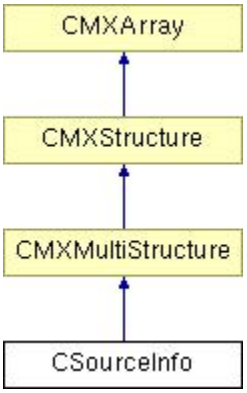

Figure 8. Inheritance diagram for **CSourceInfo**.

The **CSourceInfo** class encapsulates the SPEBE SourceInfo structure.

USE CASES:

- Inherited use cases from **CMXMultiStructure**
- Create structure by loading from a list of specified file names

4.8.1 Constructor and Destructor Documentation

CSourceInfo(double *height*)

This function creates a single source with only the height (Z coordinate) field.

Parameters:

- *height* – the height (Z coordinate) of the single element

CSourceInfo(CSourceInfo * *sourceInfo*)

This function creates a single source from a deep copy of an existing single-element. **CSourceInfo**.

Parameters:

- *sourceInfo* – the existing **CSourceInfo** object

CSourceInfo(const char * *fileName*)

This function creates a single source from file.

Parameters:

- *fileName* – the name of the Matlab *.mat* SourceInfo file describing the source

```
CSourceInfo    ( int           numSources,
                 const char * fileNames[]
               )
```

This function creates multiple sources from a collection of file.

Parameters:

- *numSources* – the number of sources
- *fileNames* – an array of size numSources, containing names of the Matlab *.mat* SourceInfo files describing the sources

```
CSourceInfo    ( int           numSources,
                 const char * fileName
               )
```

This function creates multiple, identical sources from a file.

Parameters:

- *numSources* – the number of identical sources
- *fileName* – the name of the Matlab *.mat* SourceInfo file describing the source

```
CSourceInfo ( int             numSources,
              CSourceInfo * sourceSourceInfo,
              int             sourceElement
            )
```

This function creates multiple, identical sources from a deep copy of a single element of an existing multiple-element **CSourceInfo**.

Parameters:

- *numRcvrs* – the number of identical sources to be created
- *rcvrInfo* – the existing **CSourceInfo** object
- *sourceElement* – the index of the element of the existing **CSourceInfo** object to be copied

4.8.2 Member Function Documentation

```
void LoadFile ( int             arrayElement,
                const char * fileName
              ) [virtual]
```

This function loads a single source element from file.

Parameters:

- *arrayElement* – the index of the element to be loaded from file
- *fileName* – the name of the Matlab *.mat* SourceInfo file describing the source

This function implements **CMXMultiStructure**.

```
void SetDirection ( int      arrayElement,
                    double dir
                  )
```

This function sets the direction of the specified element.

Parameters:

- *arrayElement* – index of element
- *dir* – direction (degrees)

void SetDirection(double *dir*)

This function sets the direction of all elements in the array.

Parameters:

- *dir* – direction (degrees)

4.9 CMXSingleStructure Class Reference

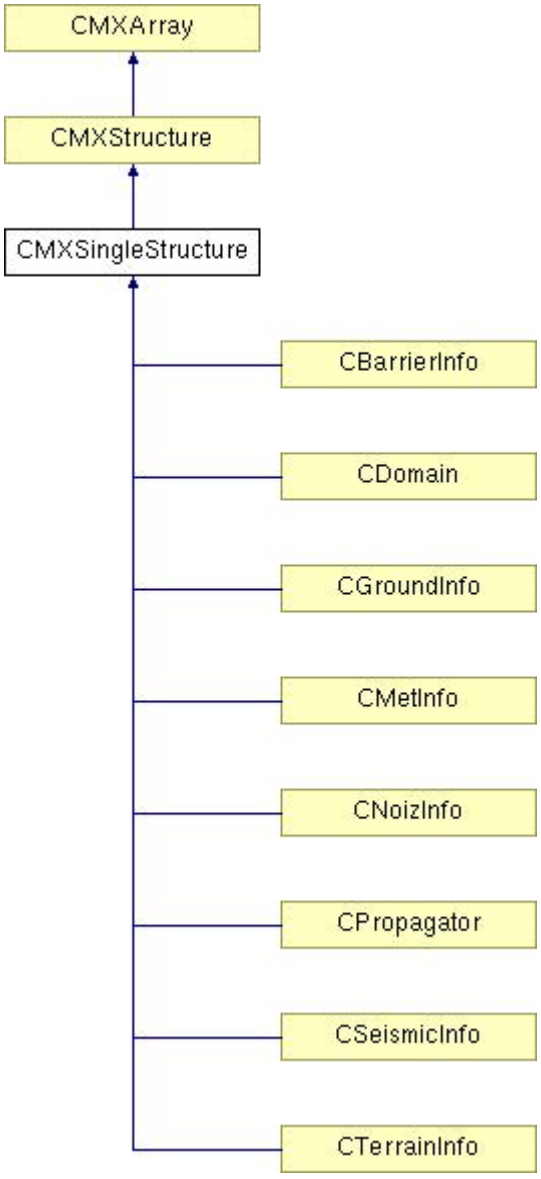

Figure 9. Inheritance diagram for **CMXSingleStructure**.

The **CMXSingleStructure** class streamlines **CMXStructure** for use with single-element Matlab structures. Arrays of Matlab structures are managed by **CMXMultiStructure**.

USE CASES:

- Create new structure of default values, then set parameters as needed.
- Create new structure from file, then set parameters as needed.

In either case, default values are also set via the appropriate SPEBE default-parameter m-file. This m-file is invoked via the SetDefault method, which is virtual in **CMXSingleStructure**, and must be overloaded by each derived class.

See also **CMXStructure**, **CMetInfo**, **CGroundInfo**, **CNoizInfo**, **CSeismicInfo**, **CTerrainInfo**, **CBarrierInfo**, **CDomain**, and **CPropagator**.

4.9.1 Member Function Documentation

double GetDouble(const char * *fieldName*)

This function returns the specified double.

Parameters:

- *fieldName* – string identifying the name of the field to be set

Returns: The value requested

CMXArray * GetField(const char * *fieldName*)

This function returns the data of a specified field.

Parameters:

- *fieldName* – string identifying the name of the desired field

Returns: A copy of the requested field.

const char * GetString(const char * *fieldName*)

This function returns a const char* of the specified field.

Warning: GetString uses a single string buffer to return a const char* string. Thus, the contents of the buffer will be overwritten upon the next call to GetString. If the returned string must persist after the next call, then copy it using strdup.

Parameters:

- *fieldName* – string identifying the name of the field to be retrieved

Returns: The const char* of the requested field

4.10 CBarrierInfo Class Reference

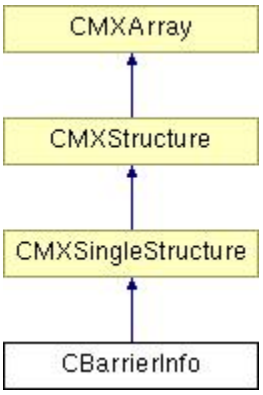

Figure 10. Inheritance diagram for **CBarrierInfo**.

The **CBarrierInfo** class encapsulates the SPEBE BarrierInfo structure.

Barriers are not currently supported by the API, so this class has no methods for specifying, manipulating, or retrieving parameters. It is provided because it is needed internally by some of the terrain and domain related m-files which are part of the compiled library.

See also **CDomain**.

NOTE: Appendix H: **CBarrierInfo** Supplemental Information has not been included in this report.

4.10.1 Constructor and Destructor Documentation

CBarrierInfo(void)

This function creates with SPEBE default parameters.

CBarrierInfo(CBarrierInfo * *theBarrierInfo*)

This function creates a deep copy of another **CBarrierInfo** object.

Parameters:

- *theBarrierInfo* – the existing **CBarrierInfo** object to copy

4.11 CDomain Class Reference

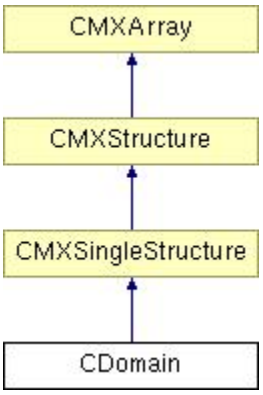

Figure 11. Inheritance diagram for **CDomain**.

The CDomain class encapsulates the SPEBE Origin, Nx, Ny, Lx, Ly, heights, xdom, and ydom structures, described in this section. Also contains **CGroundInfo**, **CTerrainInfo**, **CSeismicInfo**, and **CBarrierInfo** objects for convenience. All these can be initialized from a single call to LoadFile, using a *Terrain.mat* file.

Use this class to define the SPEBE domain, which includes the following:

- *Origin*: the latitude and longitude of the southwest corner of the computational domain
- *Extent*: the north-south (Ly) and east-west (Lx) extent of the computational domain, in meters
- *Computational grid*: the uniform grid of Nx by Ny points at which computations are performed
- *Terrain elevations*: read from a Digital Terrain Elevation Data (DTED) directory, which could be on CD or hard drive
- *Timezone*: the difference between local time and Greenwich Mean Time (GMT)
- xdom, ydom: used by several SPEBE m-files, and are generated internally from Lx, Ly, Nx, and Ny.

The **CGroundInfo**, **CTerrainInfo**, **CSeismicInfo,** and **CBarrierInfo** objects are included for several reasons:

- All are loaded from a *Terrain.mat* file.
- All are related to the characterization of the static propagation environment.
- All are generally required by propagation and/or sensor calculations, so their inclusion in a single class significantly shortens the argument list of many methods.

Although the **CMetInfo** also describes the propagation environment, it is not included for two reasons:

- It does not represent a static characterization of the environment.
- It is not loaded from a *Terrain.mat* file.

NOTE: The setters and getters (SetGroundInfo, GetSeismicInfo, etc.) for the embedded **CMXSingleStructures** GroundInfo, SeismicInfo, TerrainInfo, and BarrierInfo adhere to the following:

- The setters make a deep copy of their argument so that the embedded structure is not affected by subsequent changes to the argument.
- The getters return the actual embedded structure, so that the structure can be modified directly, rather than having to set a new structure after each change. This doesn't really violate encapsulation, since the data within the embedded structure are still encapsulated.

Internal behavioral details.

For a detailed description of the encapsulated Matlab Origin structure and miscellaneous parameters, see appendix A.

4.11.1 Constructor and Destructor Documentation

CDomain(CDomain * *domain*)

This function is a copy constructor that makes a deep copy of the argument.

Parameters:

- *domain* – the existing **CDomain** object to copy

CDomain(const char * *fileName*)

This function instantiates a **CDomain** domain object with data obtained from a *Terrain.mat* file.

Parameters:

- *fileName* – the .mat file from which to read the **CDomain** parameters

4.11.2 Member Function Documentation

CBarrierInfo * GetBarrierInfo(void)

This function returns a pointer to the internal **CBarrierInfo** object. Thus, changes made to the returned **CBarrierInfo** object will be reflected in the internal **CBarrierInfo** object. Compare this with **CDomain::SetBarrierInfo**.

Returns: A pointer to the internal **CBarrierInfo** object

double GetEasternExtent(void)

This function returns the extent of the computational domain in the x (E-W) dimension.

Returns: The eastern extent

CGroundInfo * GetGroundInfo(void)

This function returns a pointer to the internal **CGroundInfo** object. Thus, changes made to the returned **CGroundInfo** object will be reflected in the internal **CGroundInfo** object. Compare this with **CDomain::SetGroundInfo**.

Returns: A pointer to the internal **CGroundInfo** object

CMXNumeric * GetHeights(void)

This function returns the terrain elevations.

Returns: The terrain elevations as a **CMXNumeric**

double GetLatitude(void)

This function returns the latitude of the southeast corner of the computational domain.

Returns: The latitude as a signed decimal (positive north, negative south)

double GetLongitude(void)

This function returns the longitude of the southeast corner of the computational domain.

Returns: The longitude as a signed decimal (positive east, negative west)

CMXNumeric * GetLx(void)

This function returns the extent of the computational domain in the x (E–W) dimension.

Returns: The eastern extent as a **CMXNumeric**

CMXNumeric * GetLy(void)

This function returns the extent of the computational domain in the y (N–S) dimension.

Returns: The northern extent as a **CMXNumeric**

double GetNorthernExtent(void)

This function returns the extent of the computational domain in the y (N–S) dimension.

Returns: The northern extent

int GetNumX(void)

This function returns the number of computational grid points in the x (E–W) dimension.

Returns: The number of grid points returned as an integer

int GetNumY(void)

This function returns the number of computational grid points in the y (N–S) dimension.

Returns: The number of grid points returned as an integer

CMXNumeric * GetNx(void)

This function returns the number of computational grid points in the x (E–W) dimension.

Returns: The number of grid points returned as a **CMXNumeric**

CMXNumeric * GetNy(void)

This function returns the number of computational grid points in the y (N–S) dimension.

Returns: The number of grid points returned as a **CMXNumeric**

CSeismicInfo * GetSeismicInfo(void)

This function returns a pointer to the internal **CSeismicInfo** object. Thus, changes made to the returned **CSeismicInfo** object will be reflected in the internal **CSeismicInfo** object. Compare this with **CDomain::SetSeismicInfo**.

Returns: A pointer to the internal **CSeismicInfo** object

CTerrainInfo * GetTerrainInfo(void)

This function returns a pointer to the internal **CTerrainInfo** object. Thus, changes made to the returned **CTerrainInfo** object will be reflected in the internal **CTerrainInfo** object. Compare this with **CDomain::SetTerrainInfo**.

Returns: A pointer to the internal **CTerrainInfo** object

int GetTimeZone(void)

This function returns the timezone of the computational domain.

Returns: The timezone

void LoadDTEDHeights(const char * ***DTEDPath*** **)**

This function loads terrain elevations from a DTED directory, which can be on CD or hard drive. **Attention:** Terrain elevation data are not updated after calls to SetLocation, SetGridDimensions, or SetGridResolution. Thus, LoadDTEDHeights must be called *after* all of those calls.

Parameters:

- *DTEDPath* – the DTED directory

void LoadFile(const char * ***fileName*** **)** `[virtual]`

This function loads all CDomain data, including that of the embedded **CGroundInfo**, **CTerrainInfo**, **CSeismicInfo,** and **CBarrierInfo** objects, from the specified file.

Parameters:

- *fileName* – the *.mat* file from which to read the **CDomain** parameters

This function was reimplemented from **CMXSingleStructure**.

void SaveToFile(const char * ***fileName*** **)** `[virtual]`

This function saves all the encapsulated data to a single *.mat* file.

Warning: An existing file will be overwritten without warning.

Parameters:

- *fileName* – the file in which to save the encapsulated data

This function was reimplemented from **CMXArray**.

void SetBarrierInfo(CBarrierInfo * *barrierInfo*)

This function makes a deep copy of *barrierInfo* and places it in the internal **CBarrierInfo** object. Thus, any changes made to *barrierInfo* after the call to SetBarrierInfo will NOT be reflected in the internal **CBarrierInfo** object. Compare this with **CDomain::GetBarrierInfo**.

Parameters:

- *barrierInfo* – an external **CBarrierInfo** object

void SetGridDimensions(int *Nx*,
int *Ny*
)

This function sets the dimensions of the computational grid, the points at which computations are performed within the computational domain.

Parameters:

- *Nx* – the number of points in the x (E– W) coordinate
- *Ny* – the number of points in the y (N– S) coordinate

void SetGridResolution(double *xResolution*,
double *yResolution*
)

This function specifies the grid resolution in meters. This function calculates new Nx and Ny based on the specified resolution and the current values of Lx and Ly. The grid resolution is not a persistent SPEBE field, whereas Nx and Ny are. Thus, if new coordinates are set subsequent to this call, Nx and Ny will remain fixed, and the resulting resolution will change. In order to be assured of a specific resolution, this call must be made *after* coordinates have been set.

Parameters:

- *xResolution* – the distance in meters between grid points in the x, or east/west, direction
- *yResolution* – the distance in meters between grid points in ty y, or north/south, direction

void SetGroundInfo(CGroundInfo * *groundInfo*)

This function makes a deep copy of *groundInfo* and places it in the internal **CGroundInfo** object. Thus, any changes made to *groundInfo* after the call to SetGroundInfo will NOT be reflected in the internal **CGroundInfo** object. Compare this with **CDomain::GetGroundInfo**.

Parameters:

- *groundInfo* – an external **CGroundInfo** object

```
void SetLocation1   ( double dLat,
                      double dLon,
                      double dEasternExtent,
                      double dNorthernExtent,
                      int    iTimeZone
                    )
```

This function sets the coordinates of the southwest corner of the computational domain, the extents of the computational domain, and the time zone.

Parameters:

- *dLat* – the latitude of the origin, in signed decimal (+ north)
- *dLong* – the longitude of the origin, in signed decimal (+ east)
- *dEasternExtent* – the eastern (x) extent of the computational domain, in meters
- *dNorthernExtent* – the northern (y) extent of the computational domain, in meters

```
void SetLocation2   ( double lowerLeftLat,
                      double lowerLeftLon,
                      double upperRightLat,
                      double upperRightLon,
                      int    iTimeZone
                    )
```

This function sets the coordinates of the southwest and northeast corners of the computational domain and the time zone.

Parameters:

- *lowerLeftLat* – the latitude of the southwest corner, in signed decimal (+ north)
- *lowerLeftLon* – the longitude of the southwest corner, in signed decimal (+ east)
- *upperRightLat* – the latitude of the northeast corner, in signed decimal (+ north)
- *upperRightLon* – the longitude of the northeast corner, in signed decimal (+ east)

```
void SetSeismicInfo( CSeismicInfo * seismicInfo )
```

This function makes a deep copy of ***CSeismicInfo*** and places it in the internal **CSeismicInfo** object. Thus, any changes made to *seismicInfo* after the call to SetSeismicInfo will NOT be reflected in the internal **CSeismicInfo** object. Compare this with **CDomain::GetSeismicInfo**.

Parameters:

- *seismicInfo* – an external **CSeismicInfo** object

void SetTerrainInfo(CTerrainInfo * *terrainInfo*)

This function makes a deep copy of *terrainInfo* and places it in the internal **CTerrainInfo** object. Thus, any changes made to *terrainInfo* after the call to SetTerrainInfo will NOT be reflected in the internal **CTerrainInfo** object. Compare this with **CDomain::GetTerrainInfo**.

Parameters:

- *terrainInfo* – an external **CTerrainInfo** object

4.12 CGroundInfo Class Reference

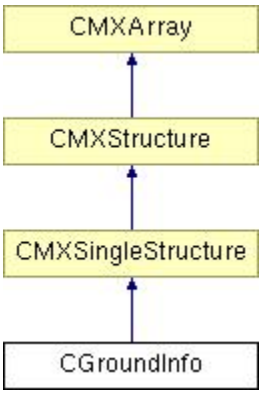

Figure 12. Inheritance diagram for **CGroundInfo**.

The **CGroundInfo** class encapsulates the SPEBE GroundInfo structure, which contains a number of parameters characterizing the ground with regard to its acoustic impedance, effect on wind, and thermal absorption and radiation characteristics. However, these parameters are not manipulated within the scope of the SPEBE API, but instead are specified via a collection of files describing various common ground types, including

- Urban,
- Suburban,
- Asphalt,
- Gravel,
- Sand,
- Brush,
- Forest,
- Short grass,
- Long grass,

- Open water,
- Ice, and
- Snow.

CGroundInfo objects are created using the constructor **CGroundInfo(const char *fileName)**, which takes a Matlab data file containing a GroundInfo structure. The API includes the following files: *Urban.mat*, *Suburban.mat*, *Asphalt.mat*, *Gravel.mat*, *Sand.mat*, *Brush.mat*, *Forest.mat*, *ShortGrass.mat*, *LongGrass.mat*, *OpenWater.mat*, *Ice.mat*, and *Snow.mat*.

Internal behavioral details.

Users can also provide their own files, which must be created within Matlab. For a detailed description of the encapsulated Matlab GroundInfo structure, see appendix D.

4.12.1 Constructor and Destructor Documentation

CGroundInfo(void)

This function is created with SPEBE default parameters.

CGroundInfo(CGroundInfo * *theGroundInfo*)

This function is created with a deep copy of another **CGroundInfo** object.

Parameters:

- *theGroundInfo* – the existing **CGroundInfo** object to copy

CGroundInfo(const char * *fileName*)

This function is created with parameters specified in a *.mat* GroundInfo file.

Parameters:

- *fileName* – the name of a *.mat* file with GroundInfo data

4.13 CMetInfo Class Reference

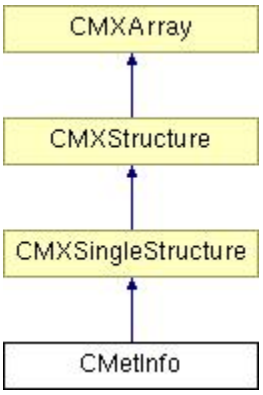

Figure 13. Inheritance diagram for **CMetInfo**.

The **CMetInfo** class encapsulates the SPEBE MetInfo structure. It includes methods to specify meteorological parameters in several different modes:

- Directly specify a pre-defined refractive profile, using the enumerated type **RefractiveProfile**.
- Specify one of a predefined set of conditions, using the enumerated type **PredefinedMet**, combining one each from the following:
  - sunny day, night, or overcast
  - light wind, moderate wind, or strong wind
- Specify a set of surface parameters, using **SetSurfaceParameters**.
- Specify a set of similarity parameters, using **SetScalingParameters**.

Internal behavioral details.

For a detailed description of the encapsulated Matlab MetInfo structure, see appendix C.

4.13.1 Member Enumeration Documentation

enum PredefinedMet

This function is for selection of a predefined meteorological case.

Values: SunnyLightWind, SunnyModerateWind, SunnyStrongWind, NightLightWind, NightModerateWind, NightStrongWind, OvercastLightWind, OvercastModerateWind, OvercastStrongWind

enum RefractiveProfile

This function is for selection of a specific refractive profile type.

Values: NoRefraction, MildUpwardRefraction, StrongUpwardRefraction, DownwardRefraction, ShallowInversion, DeepInversion

4.13.2 Constructor and Destructor Documentation

CMetInfo(void)

This function is constructed using default values for all fields.

CMetInfo(CMetInfo * *mInfo*)

This function is constructed using a deep copy of an existing **CMetInfo** object.

Parameters:

- *mInfo* – the existing object to copy

CMetInfo(const char * *fileName*)

This function is constructed with values determined by the specified MetInfo type *.mat* file.

Parameters:

- *fileName* – the name of a *.mat* file containing MetInfo values

4.13.2 Member Function Documentation

double GetT0(void)

This function returns the surface temperature T0.

Returns: The surface temperature

```
void SetPredefinedMeteorology( PredefinedMet  metCase,
                               double         dWindDir,
                               double         dTemp,
                               double         dTempHeight,
                               double         dRelHumid,
                               double         dRelHumHeight,
                               CGroundInfo *  ground
                             )
```

This function is set from a predefined meteorological condition, chosen from the enumerated type PredefinedMet, with specified surface temperature and relative humidity.

Options for the profiles are the following:

- Mostly sunny day, light wind
- Mostly sunny day, moderate wind
- Mostly sunny day, strong wind
- Mostly clear night, light wind
- Mostly clear night, moderate wind
- Mostly clear night, strong wind
- Overcast (day or night), light wind
- Overcast (day or night), moderate wind
- Overcast (day or night), strong wind

Parameters:

- *metCase* – the predefined profile, chosen from PredefinedMet
- *dWindDir* – the wind direction
- *dTemp* – the near-surface temperature, in degrees Celsius
- *dTempHeight* – the height of the near-surface temperature measurement or forecast, in meters
- *dRelHumid* – the near-surface relative humidity, in percent (0% to 100%)
- *dRelHumHeight* – the height of the near-surface relative humidity measurement or forecast, in meters
- *ground* – a **CGroundInfo** object (needed for the roughness height and displacement height)

```
void SetRefractiveProfileType( RefractiveProfile refCase,
                               double            T0,
                               double            RH
                             )
```

This function sets a specified refractive profile, chosen from the enumerated type RefractiveProfile, with specified surface temperature and relative humidity.

Options for the profiles are the following:

- No refraction
- Mild upward refraction
- Strong upward refraction
- Downward refraction
- Shallow inversion
- Deep inversion

Parameters:

- *refCase* – the refractive profile type, chosen from RefractiveProfile
- *T0* – the surface temperature, in degrees Celsius
- *RH* – the relative humidity, in percent (0% to 100%)

```
void SetScalingParameters( double          dWindDir,
                           double          dSurfTemp,
                           double          dTempHeight,
                           double          dRelHumid,
                           double          dRelHeight,
                           double          dFricVel,
                           double          dTempScale,
                           double          dSpecHumScale,
                           double          dInvHgt,
                           CGroundInfo *   ground
                         )
```

This function sets a collection of surface similarity scaling parameters, including

- wind direction,
- temperature,
- humidity,

- friction velocity u_*,
- temperature scale T_*,
- specific humidity scale q_*,
- inversion height, and
- roughness element and displacement height.

Parameters:

- *dWindDir* – the wind direction
- *dTemp* – the near-surface temperature, in degrees Celsius
- *dTempHeight* – the height of the near-surface temperature measurement or forecast, in meters
- *dRelHumid* – the near-surface relative humidity, in percent (0% to 100%)
- *dRelHumHeight* – the height of the near-surface relative humidity measurement or forecast, in meters
- *dFricVel* – the friction velocity, in meters/second
- *dTempScale* – the temperature scale, in degrees Celsius
- *dSpecHumScale* – the specific humidity scale
- *dInvHgt* - the height of the first inversion, in meters
- *ground* - a **CGroundInfo** object (needed for the roughness element and displacement height)

```
void SetSurfaceParameters( double      dWindDir,
                           double      dWindSPeed,
                           double      dWindHt,
                           double      dSurfTemp,
                           double      dTempHeight,
                           double      dRelHumid,
                           double      dRelHeight,
                           double      dInvHgt,
                           double      cloudCover[3],
                           int         timeOfDay,
                           int         dayOfYear,
                           int         year,
                           CDomain *   domain
                         )
```

This function sets a collection of surface parameters, including

- wind speed and direction,
- temperature,
- humidity,
- inversion height,
- cloud cover,
- time of day, day of year, and year, and
- geographic coordinates.

Parameters:

- *dWindDir* – the wind direction
- *dTemp* – the near-surface temperature, in degrees Celsius
- *dTempHeight* – the height of the near-surface temperature measurement or forecast, in meters
- *dRelHumid* – the near-surface relative humidity, in percent (0% to 100%)
- *dRelHumHeight* – the height of the near-surface relative humidity measurement or forecast, in meters
- *dInvHgt* – the height of the first inversion, in meters
- *cloudCover* – a three-element decimal array of cloud-cover fraction at low, medium, and high elevation
- *timeOfDay* – time of day, in hours since midnight
- *dayOfYear* – Julian date
- *year* – four-digit year
- *domain* – a **CDomain** object (needed for geographic location)

4.14 CNoizInfo Class Reference

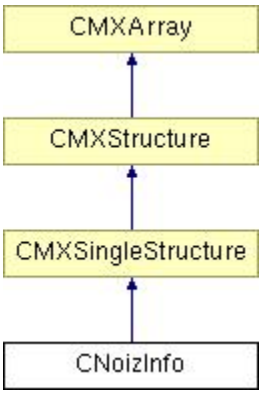

Figure 14. Inheritance diagram for **CNoizInfo**.

The **CNoizInfo** class Encapsulates the SPEBE NoizInfo structure, which contains three fields—the acoustic spectrum, the seismic spectrum, and a flag—to indicate whether or not to include acoustic wind noise. The acoustic and seismic spectra are, in turn, described by structures defining the spectral bands in terms of lower and upper frequency, slope, and loudness.

The NoizInfo parameters are not manipulated directly by the user, and are defined by the specification of separate acoustic and seismic noise files. Several files of each are provided as part of SPEBE, or new ones can be generated, via the SPEBE GUI, or directly in Matlab. The parameters may also be set via a single file containing both the acoustic and seismic structures as well as the IncWind field.

Internal behavioral details.

For a detailed description of the encapsulated Matlab NoizInfo structures, see appendix F.

4.14.1 Constructor and Destructor Documentation

CNoizInfo(void)

This function is created with SPEBE default parameters.

CNoizInfo(CNoizInfo * *theNoizInfo*)

This function is created with a deep copy of another **CNoizInfo** object.

Parameters:

- *theNoizInfo* – the existing **CNoizInfo** object to copy

CNoizInfo(const char * *fileName*)

This function is created using a NoiseInfo file, which must contain both acoustic and seismic noise levels, as well as IncWind. This will not work with an acoustic noise file or a seismic noise file.

Parameters:

- *fileName* – the name of the file containing both acoustic and seismic structures

4.14.2 Member Function Documentation

void SetAcousticLevel(const char * *fileName*)

This function sets the acoustic background characteristics by loading parameters from a *.mat* acoustic noise file.

Parameters:

- *fileName* – the name of the file containing the acoustic parameters

void SetSeismicLevel(const char * *fileName*)

This function sets the seismic background characteristics by loading parameters from a *.mat* seismic noise file.

Parameters:

- *fileName* – the name of the file containing the seismic parameters

4.15 CPropagator Class Reference

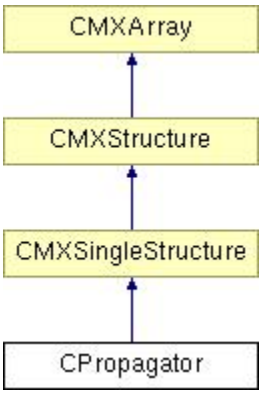

Figure 15. Inheritance diagram for **CPropagator**.

The **CPropagator** class encapsulates the SPEBE PropInfo structure. This also manages acoustic and seismic PropTab structures internally and includes a **CNoizInfo** object for convenience. All these can be initialized with from a single call to LoadFile from a *Prop.mat* file.

The **CNoizInfo** object is included for two reasons:

- It is loaded from the same *Prop.mat* file as PropInfo and the PropTabs.
- It is generally required along with the propagation tables and PropInfo for sensor calculations.

NOTE: Although the **CNoizInfo** object is packaged with the **CPropagator**, it is not used in propagation table calculations. Thus, the **CNoizInfo** object can be changed independently of propagation table generation.

Setters and getters for the embedded **CMXSingleStructure** NoizInfo adhere to the following:

- The setters make a deep copy of their argument so that the embedded structure is not affected by subsequent changes to the argument.
- The getters return the actual embedded structure, so that the structure can be modified directly, rather than having to set a new structure after each change. This doesn't really violate encapsulation, since the data within the embedded structure are still encapsulated.

Internal behavioral details.

For a detailed description of the encapsulated Matlab PropInfo, AcsPropTab, and SeisPropTab structures, see appendix B.

4.15.1 Member Enumeration Documentation

enum AcsPropModel

This function is for selection of a specific acoustic propagation model.

Values:

- SPHER_SPREAD – spherical spreading with perfectly reflecting ground
- IMPED_PLANE – spherical spreading with finite impedance ground
- IMPED_PLANE_TURB – spherical spreading with finite impedance ground and empirical adjustment for turbulent scattering into interference nulls
- RAY_TRACE – ray tracing
- FFP – Fast Field Program
- CPE – finite difference Parabolic Equation
- SCAPE – Green's function (split-step) Parabolic Equation
- LOAD_TABLE_ACS – load from an acoustic propagation table

enum SeisPropModel

This function is for selection of a specific seismic propagation model.

Values:

- CYLIN_SPREAD – cylindrical spreading
- HARVEY_MODE_SUM – Harvey mode summation
- LOAD_TABLE_SEIS – load from a seismic propagation table

4.15.2 Constructor and Destructor Documentation

CPropagator(CPropagator * *thePropagator*)

This function is a copy constructor.

Parameters:

- *domain* – the existing **CDomain** object to copy

CPropagator(const char * *fileName*)

This function loads all **CPropagator** data, including that of the embedded **CNoizInfo** object, from the specified file.

Parameters:

- *fileName* – the *.mat* file from which to read the **CPropagator** parameters

4.15.3 Member Function Documentation

void GenerateAcousticPropagationTable(double *sourceHeight,
double *receiverHeight,
CMetInfo * *metInfo,
CDomain * *domain,
const char * *fileName*
)

This function generates an acoustic propagation table using the propagation model specified by SetAcsPropMod and saves it in the specified file.

See also **CMetInfo** and **CDomainInfo**.

Parameters:

- *dSourceHeight* – the height of the source
- *dReceiverHeight* – the height of the receiver
- *metInfo* – pointer to the **CMetInfo** class holding meteorological parameters and profiles
- *domainInfo* – pointer to the **CDomain** class holding the domain parameters
- *fileName* – the string holding the file name

**void GenerateSeismicPropagationTable(double *sourceHeight,*
double *receiverHeight,*
CDomain * *domain,*
const char * *fileName*
)**

This function generates a seismic propagation table using the propagation model specified by SetSeisPropMod and saves it in the specified file.

See also **CMetInfo** and **CDomainInfo**.

Parameters:

- *dSourceHeight* – the height of the source
- *dReceiverHeight* – the height of the receiver
- *metInfo* – pointer to the **CMetInfo** class holding meteorological parameters and profiles
- *domainInfo* – pointer to the **CDomain** class holding the domain parameters
- *fileName* – the string holding the file name

CNoizInfo * GetNoizInfo(void)

This function returns a pointer to the internal **CNoizInfo** object. Thus, changes made to the returned **CGNoizInfo** object will be reflected in the internal **CNoizInfo** object. Compare this with **CPropagator::SetNoizInfo**.

Returns: A pointer to the internal **CNoizInfo** object

void LoadFile(const char * *fileName*) `[virtual]`

This function loads all **CPropagator** data, including that of the embedded **CNoizInfo** object, from the specified file.

Parameters:

- *fileName* – the *.mat* file from which to read the **CPropagator** parameters

This function was reimplemented from **CMXSingleStructure**.

void SaveToFile(const char * *fileName*) [virtual]

This function saves all the encapsulated data to a single *.mat* file.

Warning: An existing file will be overwritten without warning.

Parameters:

- *fileName* – the file in which to save the encapsulated data

This function was reimplemented from **CMXArray**.

void SetAcsFileName(const char * *fileName*)

This function sets the file name to be used for acoustic propagation table.

Parameters:

- *fileName* – the string holding the file name

void SetAcsPropMod(AcsPropModel *acsPropModel*)

This function sets the acoustic propagation model to be used. It is selected from the enumerated type acsPropModel.

Parameters:

- *acsPropModel* – the enumerated type which specifies the acoustic propagation model to be used

void SetNoizInfo(CNoizInfo * *noizInfo*)

This function makes a deep copy of *noizInfo* and places it in the internal NoizInfo object. Thus, any changes made to *groundInfo* after the call to SetNoizInfo will NOT be reflected in the internal **CNoizInfo** object. Compare this with **CPropagator::GetNoizInfo**.

Parameters:

- *noizInfo* – an external noizInfo object

void SetSeisFileName(const char * *fileName*)

This function sets the file name to be used for seismic propagation table.

Parameters:

- *fileName* – the string holding the file name

void SetSeisPropMod(SeisPropModel *seisPropModel*)

This function sets the seismic propagation model to be used. It is selected from the enumerated type seisPropModel.

Parameters:

- *seisPropModel* – the enumerated type which specifies the seismic propagation model to be used

4.16 CSeismicInfo Class Reference

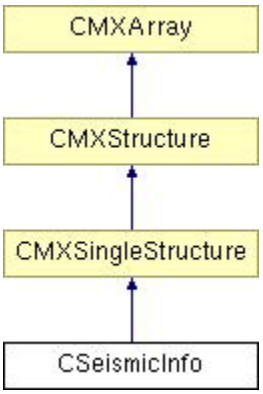

Figure 16. Inheritance diagram for **CSeismicInfo**.

The **CSeismicInfo** class encapsulates the SPEBE SeismicInfo structure, which defines vertical profiles of P and S-wave velocity and attenuation as well as density.

The SeismicInfo parameters are not manipulated directly by the user and are defined by the specification of a Matlab .mat SeismicInfo file. Several files are provided as part of SPEBE, and new ones can be generated directly in Matlab.

Internal behavioral details.

For a detailed description of the encapsulated Matlab SeismicInfo structure see appendix E.

4.16.1 Constructor and Destructor Documentation

CSeismicInfo(void)

This function is created with SPEBE default parameters.

CSeismicInfo(CSeismicInfo * *theSeismicInfo*)

This function is created using a deep copy of another **CSeismicInfo** object.

Parameters:

- *theSeismicInfo* – the existing **CSeismicInfo** object to copy

CSeismicInfo(const char * *fileName*)

This function is created using a SeismicInfo file.

Parameters:

- *fileName* – the name of the file containing SeismicInfo profiles

4.17 CTerrainInfo Class Reference

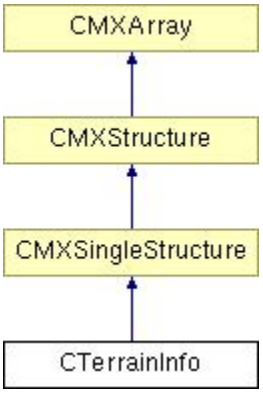

Figure 17. Inheritance diagram for **CTerrainInfo**.

The **CTerrainInfo** class encapsulates the SPEBE TerrainInfo structure, which contains parameters characterizing random terrain elevations. Random terrain is not currently supported by the API, so this class has no methods for specifying, manipulating, or retrieving parameters. It is provided because it is needed internally by some of the terrain and domain related m-files, which are part of the compiled library.

See also **CDomain**.

NOTE: Flat terrain and digital terrain elevation data are managed by the **CDomain** class.

Internal behavioral details.

For a detailed description of the encapsulated Matlab TerrainInfo structure, see appendix G.

4.17.1 Constructor and Destructor Documentation

CTerrainInfo(void)

This function is created with SPEBE default parameters.

CTerrainInfo(CTerrainInfo * *theTerrainInfo*)

This function is created using a deep copy of another **CTerrainInfo** object.

Parameters:

- *theTerrainInfo* – the existing **CTerrainInfo** object to copy

4.18 PerformanceCalculator Class Reference

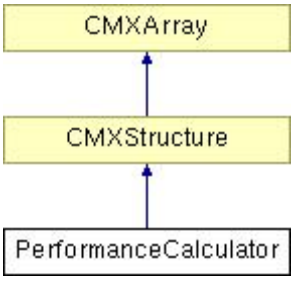

Figure 18. Inheritance diagram for **PerformanceCalculator**.

The **PerformanceCalculator** class performs the sensor performance calculations of SPEBE.

SPEBE sensor calculations operate in one of two modes:

1. Receiver-centric:
   - User has selected one or more active receivers (by clicking on them on the map).
   - No sources are active.
   - User selects a particular source type from calculation pull-down.
2. Source-centric:
   - User has selected one or more active sources.
   - No receivers are active.
   - User selects a particular receiver type from calculation pull-down.

The **PerformanceCalculator** mimics this behavior by allowing users to work in receiver-centric or source-centric mode, depending on the value of the constructor parameter **Recip**. (Note that this is static; if the **PerformanceCalculator** is instantiated in receiver-centric mode, then it cannot be changed to source-centric. A new **PerformanceCalculator** must be instantiated for that purpose.) The mode-specific behavior of the class is as follows:

Recip = RcvrCentric

- User activates/deactivates particular receivers within the **CRcvrInfo** via the Activate and Deactivate methods.
- User specifies which source within the **CSourcInfo** to use for a particular calculation by specifying its index via the argument *srIndex* in the particular calculation method.

Recip = SourceCentric

- User activates/deactivates particular sources within the **CSourceInfo** via the Activate and Deactivate methods.
- User specifies which receiver within the **CRcvrInfo** to use for a particular calculation by specifying its index via the argument *srIndex* in the particular calculation method.

If no file names are set using SetAcsPropTableFileName or SetSeisPropTableFileName, then the file name in the associated PropTab structure in the **CPropagator** will be used for all associated calculations. If specific filenames are set using SetAcsPropTableFileName or SetSeisPropTableFileName, then those file names will be used instead. Note that file names can be set for all or only specific sources or sensors. Any file name not specifically set will default to the name stored in the associated PropTab structure.

NOTE: The constructor accepts a **CRcvrInfo**, **CSourceInfo**, **CMetInfo**, **CDomain**, and **CPropagator** object, and also a **Recip** argument. PerformanceCalculators work on a *snapshot* of these arguments, meaning they are copied into the **PerformanceCalculator** where they remain static. Any changes to the structures originally passed as arguments will not be reflected in the **PerformanceCalculator**. If changes are made to the domain, met, sources, receivers, etc., and sensor calculations are required for the updated objects then a new **PerformanceCalculator** must be instantiated.

See also **ResultGrid**.

Internal behavioral details.

For a detailed description of the compiled m-files called and associated parameters, see appendix K.

4.18.1 Member Enumeration Documentation

enum Recip

This function sets the source/sensor configuration mode to source-centric or receiver-centric.

Values: SourceCentric, RcvrCentric

4.18.2 Constructor and Destructor Documentation

PerformanceCalculator(CRcvrInfo * ***rcvrInfo,***
CSourceInfo * ***sourceInfo,***
CMetInfo * ***metInfo,***
CDomain * ***domain,***
CPropagator * ***propagator,***
Recip ***recip***
)

This function instantiates a new **PerformanceCalculator**, for the specified parameters, in the specified mode.

Parameters:

- *rcvrInfo* – a **CRcvrInfo** object defining the receiver(s) to be used for the performance calculations
- *sourceInfo* – a **CSourceInfo** object defining the sources(s) to be used for the performance calculations
- *metInfo* – a **CMetInfo** object defining the meteorological parameters to be used for the performance calculations
- *domain* – a **CRcvrInfo** object defining the receiver(s) to be used for the performance calculations
- *propagator* – a **CDomainInfo** object defining the domain to be used for the performance calculations
- *recip* – specify source-centric or receiver-centric mode

4.18.3 Member Function Documentation

void Activate(int *srIndex*)

This function is activates the specified source (in source-centric mode) or receiver (in receiver-centric mode), depending on the mode.

Parameters:

- *srIndex* – the source or receiver to be activated

void Deactivate(int *srIndex*)

This function deactivates the specified source (in source-centric mode) or receiver (in receiver-centric mode), depending on the mode.

Parameters:

- *srIndex* – the source or receiver to be activated

const char * GetAcsPropTableFileName(int *sourceIndex*,
int *rcvrIndex*
)

This function gets the file name to be used for the specified acoustic propagation table.

Parameters:

- *sourceIndex* – the specified source
- *rcvrIndex* – the specified receiver

Returns: The name to be used for the specified acoustic propagation table

int GetNumReceivers(void)

This function indicates the number of receivers defined, independent of mode or activation state.

Returns: The number of receivers

int GetNumSoruces(void)

This function indicates the number of sources defined, independent of mode or activation state.

Returns: The number of sources

ResultGrid * GetPD (int *sourceIndex*,
int *rcvrIndex*
)

This function returns the Probability of Detection (PD) grid for the specified source and receiver, regardless of active or inactive states.

Parameters:

- *sourceIndex* – the requested source
- *rcvrIndex* – the requested receiver

Returns: A **ResultGrid** object containing the requested detection probability data

ResultGrid * GetPD(int *srIndex*)

This function returns the PD grid for the specified sources and receivers, depending on the mode: in ReceiverCentric mode, srIndex specifies the requested source, and all active receivers are used; in SourceCentric mode, srIndex specifies the requested receiver, and all active sources are used.

Parameters:

- *srIndex* – the desired source, when in ReceiverCentric mode; the desired receiver, when in SourcCentric mode

Returns: A **ResultGrid** object containing the requested detection probability data

const char * GetSeisPropTableFileName(int *sourceIndex*,
int *rcvrIndex*
)

This function gets the file name to be used for the specified seismic propagation table.

Parameters:

- *sourceIndex* – the specified source
- *rcvrIndex* - the specified receiver

Returns: The name to be used for the specified seismic propagation table

bool IsActive(int *srIndex*)

This function indicates whether the specified source (in source-centric mode) or receiver (in receiver-centric mode) is active, depending on the mode.

Returns: The activation state of the specified source or receiver

```
void SetAcsPropTableFileName( int          sourceIndex,
                              int          rcvrIndex,
                              const char * fileName
                            )
```

This function sets the file name to be used for the specified acoustic propagation table.

Parameters:

- *sourceIndex* – the specified source
- *rcvrIndex* – the specified receiver
- *fileName* – the name to be used for the specified acoustic propagation table

```
void SetSeisPropTableFileName( int          sourceIndex,
                               int          rcvrIndex,
                               const char * fileName
                             )
```

This function sets the file name to be used for the specified seismic propagation table.

Parameters:

- *sourceIndex* – the specified source
- *rcvrIndex* – the specified receiver
- *fileName* – the name to be used for the specified seismic propagation table

5. Miscellaneous Functions

void SPEBEInitialize(void) throw (...)

Initializes the dll. Call this function once, at the beginning of the application using the dll, before calling any method or function in the API. Same as SPEBEInitialize(false).

See also **SPEBETerminate**.

void SPEBEInitialize(bool) throw (...)

Initializes the dll. Call this function once, at the beginning of the application using the dll, before calling any method or function in the API.

See also **SPEBETerminate**.

Parameters:

- *jni* – true if the dll will be used for native code in a Java application

void SPEBETerminate(void)

Closes the dll. Call this function once, at the end of the application using the dll, when there will be no further calls to methods or functions of the API.

6. Conclusion

This API has been used as the basis for a sensor planning decision aid library, which was then incorporated into an existing GUI-based sensor placement planning tool. It has also been used in an acoustic sensor performance web services demonstration. In all cases, it proved to meet the design goals of providing all the required acoustic sensor performance functionality of SPEBE, with the flexibility to incorporate that functionality into other GUI and non-GUI based systems, and without the need to deal with compiled Matlab structures or functions.

Currently, it does not include all of the functionally available in Matlab SPEBE. Many parameters and calculations that could be provided have not yet been made available through the interface. This is a relatively simple matter of adding additional methods to the existing classes in order to set and retrieve additional fields from the embedded mxArray structures, as well as invoke additional compiled m-files.

However, all the basic tools are in place to manipulate these fields and invoke these m-files, so that no additional modifications are necessary to the lower-level classes. Additional methods can be added to the high-level classes as the need arises, using the existing low-level method calls. Thus, all that is needed is familiarity with the SPEBE architecture and parameters and sufficient understanding of the necessary calculations to know the appropriate values of the parameters and the required sequence of calculations in order to produce valid results.

This was done in some cases as the sensor planning library and web services demonstration were developed, and worked quite well. The additional capabilities were provided through the addition of methods to the high-level classes, requiring only the knowledge of acoustic calculations and SPEBE capabilities, but without consideration of the nature of the compiled Matlab infrastructure.

Thus, the API appears to function as desired and should be a valuable tool for the incorporation of acoustic sensor performance modeling into Army systems.

References

1. Wilson D. K.; Norris, D. E. *Statistical distributions of acoustic signals propagated through the atmosphere,* 1998 Meeting of the IRIS Specialty Group on Acoustic and Seismic Sensing, Laurel, MD, 1999.

2. Wilson, D. K. *A Prototype Acoustic Battlefield Decision Aid Incorporating Atmospheric Effects and Arbitrary Sensor Layouts*; ARL-TR-1708; U.S. Army Research Laboratory: Adelphi, MD, 1998.

3. Wilson, D. K. *Performance Evaluator for Battlefield Environments (SPEBE) Tutorial*; RDC/CRREL TR-06-12; Cold Regions Research and Engineering Laboratory: Hanover, NH, June 2006.

4. Wilson, D. K., Moran, M. L.; Anderson, T. S. Development of a combined acoustic/seismic tactical decision aid. *Proceedings of the Military Sensing Symposia Specialty Group on Battlefield Acoustics and Seismics*, Laurel, MD, 2002.

5. Wilson, D. K.; Kalb, J. T.; Srour, N.; Pham, T; Sadler, B. M. *Sensor Algorithm Interface and Performance Simulations in an Acoustical Battlefield Decision Aid*; ARL-TR-2860; U.S. Army Research Laboratory: Adelphi, MD, 2002.

6. Wilson, D. K.; Nguyen, V. A.; Srour, N.; Noble, J. *Sound Exposure Calculations for Transient Events and Other Improvements to an Acoustical Tactical Decision Aid*; ARL-TR-2757; U.S. Army Research Laboratory: Adelphi, MD, 2002.

7. Wilson, D. K.; Noble, B. M.; Pham, T. *Simulation of detection and beamforming with acoustical ground sensors,* SPIE's 17th International Symposium on AeroSense, Orlando, FL, 2002.

8. Wilson, D. K.; Noble, J. M.; Van Aartsen; B. M., Szeto, G. L. *Battlefield decision aid for acoustical ground sensors with interface to meteorological data sources,* SPIE's 15th International Symposium on AeroSense, Orlando, FL, 2001.

9. Wilson, D. K.; Szeto G. L., Sadler B. M.; Adams, R. N., Srour, N. *Propagation and array performance modeling for acoustic tracking of cruise missiles,* 1999 Meeting of the IRIS Specialty Group on Acoustic and Seismic Sensing, Laurel, MD, 2000.

10. Wilson, D. K.; Szeto, G. L. *Reference Guide for the Acoustic Battlefield Aid (ABFA) Version 2*; ARL-TR-2159; U.S. Army Research Laboratory: Adelphi, MD, 2000.

11. Wilson, D. K.; Szeto, G. L.; Van Aartsen, B. H.; Noble, J. M. *A prototype acoustic battlefield decision aid with interfaces to meteorological data sources and a target database*, 2000 Meeting of the MSS Specialty Group on Battlefield Acoustic and Seismic Sensing, Laurel, MD, 2000.

INTENTIONALLY LEFT BLANK.

Appendix A. CDomain Supplemental Information

Tables A-1 and A-2 include the origin and miscellaneous fields encapsulated by **CDomain**.

Table A-1. Origin fields.

Field Name	Description	Type or Specified Values
lat	latitude of southwest corner of domain	signed decimal: + north, - south
long	longitude of southwest corner of domain	signed decimal: + east, - west
TimeZone	difference between local time and Universal Time Coordinated (UTC)	integer between 0 and 23

Table A-2. Miscellaneous domain parameters.

Field Name	Description	Type or Specified Values
Lx	Extent in x direction (km)	integer
Ly	Extent in y direction (km)	integer
Nx	Number of computational grid points in x direction	integer
Ny	Number of computational grid points in y direction	integer
xdom	E-W computational grid coordinates in km	real array of size Nx
ydom	N-S computational grid coordinates in km	real array of size Ny
heights	terrain elevation grid	real array of size Nx X Ny

INTENTIONALLY LEFT BLANK.

Appendix B. CPropagator Supplemental Information

Tables B-1 and B-2 include the **PropInfo** and **PropTab** fields encapsulated by **CPropagator**.

Table B-1. PropInfo fields.

Field Name	Description	Type or Specified Values
AcsPropMod	Acoustic Propagation Model	'SpherSpread' spherical spreading, infinite ground impedance 'ImpedPlane' spherical spreading, finite ground impedance 'ImpedPlaneTurb' spherical spreading, finite ground impedance with turbulence 'RayTrace' ray tracing model 'FFP' fast field program 'CNPE' Crank-Nicholson Parabolic Equation (wide-angle, finite-difference) 'SCAPE' Green's Function Parabolic Equation (narrow-angle, split-step) 'LoadTable' load from propagation table
SeisPropMod	Seismic Propagation Model	'CylinSpread' cylindrical spreading 'HarveyModeSum' Harvey mode summation 'LoadTable' load from propagation table
IncTer	Terrain diffraction calculation	0 – don't include terrain diffraction 1 – include terrain diffraction
IncCurv	Earth curvature diffraction calculation	0 – don't include earth curvature 1 – include earth curvature
IncAbsorp	include acoustic atmospheric absorption	0 – don't include atmospheric absorption 1 – include atmospheric absorption
IncSeisAbsorp	include seismic absorption	0 – don't include seismic absorption 1 – include seismic absorption
PartialSat	partial saturation	0 – don't include partial saturation 1 – include partial saturation
PartialCoh	partial coherence	0 – don't include partial coherence 1 – include partial coherence
IncGain	include array in SNR and detection calculations	0 – don't include array 1 – include array
Weighting	Acoustic sound pressure level weighting	'unweighted' 'A-weighted' 'C-weighted'

Table B-2. PropTab fields (acoustic and seismic).

Field Name	Description	Type or Specified Values
FileName	Name to use for propagation table file	
TempFileName	Name to use for temporary propagation table file	
AutoParam		
NumAng	Number of angles	Integer > 1
MinFreq	Minimum frequency	real
MaxFreq	Maximum frequency	real
NumFreq	Number of frequencies	Integer > 0
LogFreq	Logarithmic or linear frequency distribution	0 – linear; 1- logarithmic
MinRange	Minimum range	real
MaxRange	Maximum range	real
NumRange	Number of range steps	Integer > 1
LogRange	Logarithmic or linear range distribution	0 – linear; 1- logarithmic

Appendix C. CMetInfo Supplemental Information

Table C-1 includes the MetInfo fields encapsulated by **CMetInfo**.

Table C-1. MetInfo fields.

Field Name	Description	Type or Specified Values
SpecType	meteorological input type Determines how meteorological parameters or refractive profiles are specified.	1 – not used 2 – specify refractive profile (RefCase) 3 – specify predefined met case (SimpCase) 4 – specify scaling parameters 5 – specify surface data 6 – read surface message 7 – specify vertical meteorological profiles
RefCase	refractive profile type (WeatherCaseRP) Specifies what type of refractive profile to generate. Applicable only when SpecType = 2, in which case no meteorological parameters are specified. Profiles are defined in GenProf.m.	1 – don't use predefined profiles 2 – no refraction (NR) 3 – mild upward refraction (MUR) 4 – strong upward refraction (SUR) 5 – downward refraction (DR) 6 – shallow inversion (SI) 7 – deep inversion (DI)
SimpCase	predefined meteorological case (WeatherCaseMC) Specifies a predefined set of scaling parameters. Applicable only when SpecType = 3. Parameters are set in WeatherCallbacks.m.	1 – don't use predefined cases 2 – sunny day, light wind (SL) 3 – sunny day, moderate wind (SM) 4 – sunny day, strong wind (SS) 5 – night, light wind (NL) 6 – night, moderate wind (NM) 7 – night, strong wind (NS) 8 – overcast, light wind (OL) 9 – overcast, moderate wind (OM) 10 – overcast, strong wind (OS)
ShearTurb	shear turbulence model Defined in Weathermenu.m. Used by compMu2.m and compTurb.m.	2 – none 3 – Kolmogorov 4 – Isotropic Von Karman 5 – Mann RDT
BuoyTurb	buoyancy turbulence model Defined in Weathermenu.m. Used by compMu2.m and compTurb.m.	2 – none 3 – Kolmogorov 4 – Isotropic Von Karman 5 – Hunt/Graham/Wilson
TempTurb	temperature turbulence model Defined in weathermenu.m. Used by compMu2.m and compTurb.m.	2 – none 3 – Kolmogorov 4 – Isotropic Von Karman
ustar	friction velocity (m/s)	real
Tflux	surface temperature flux (K*(m/s))	real
qflux	surface specific humidity flux (m/s)	real
T0	ambient surface air temperature (C)	real
T0_hgt	height of temperature measurement (m)	real

Table C-1. MetInfo fields (continued).

Field Name	Description	Type or Specified Values
RH	surface relative humidity (%)	real
RH_hgt	height of humidity measurement (m)	real
U0	surface reference wind speed (m/s)	real
U0_hgt	height of wind velocity measurement (m)	real
winddir	wind direction (deg)	real
zi	inversion height	real
CloudCover	fractional cloud cover, at low-, mid-, and high-	real 3-element vector
Year	calendar year	integer
DayOfYear	day of year (Jan. 1 = 0)	integer
TimeOfDay	time of day, local standard time (hours)	integer
Stations	weather stations on display	
VirtStations	virtual (forecast) weather stations on display	
StationID	ICAO ID of current station loaded	string
vx_prof	x component of vertical velocity profile (m/s)	real vector
vy_prof	y component of vertical velocity profile (m/s)	real vector
T_prof	vertical temperature profile (Kelvin)	real vector
q_prof	vertical humidity profile (relative humidity)	real vector

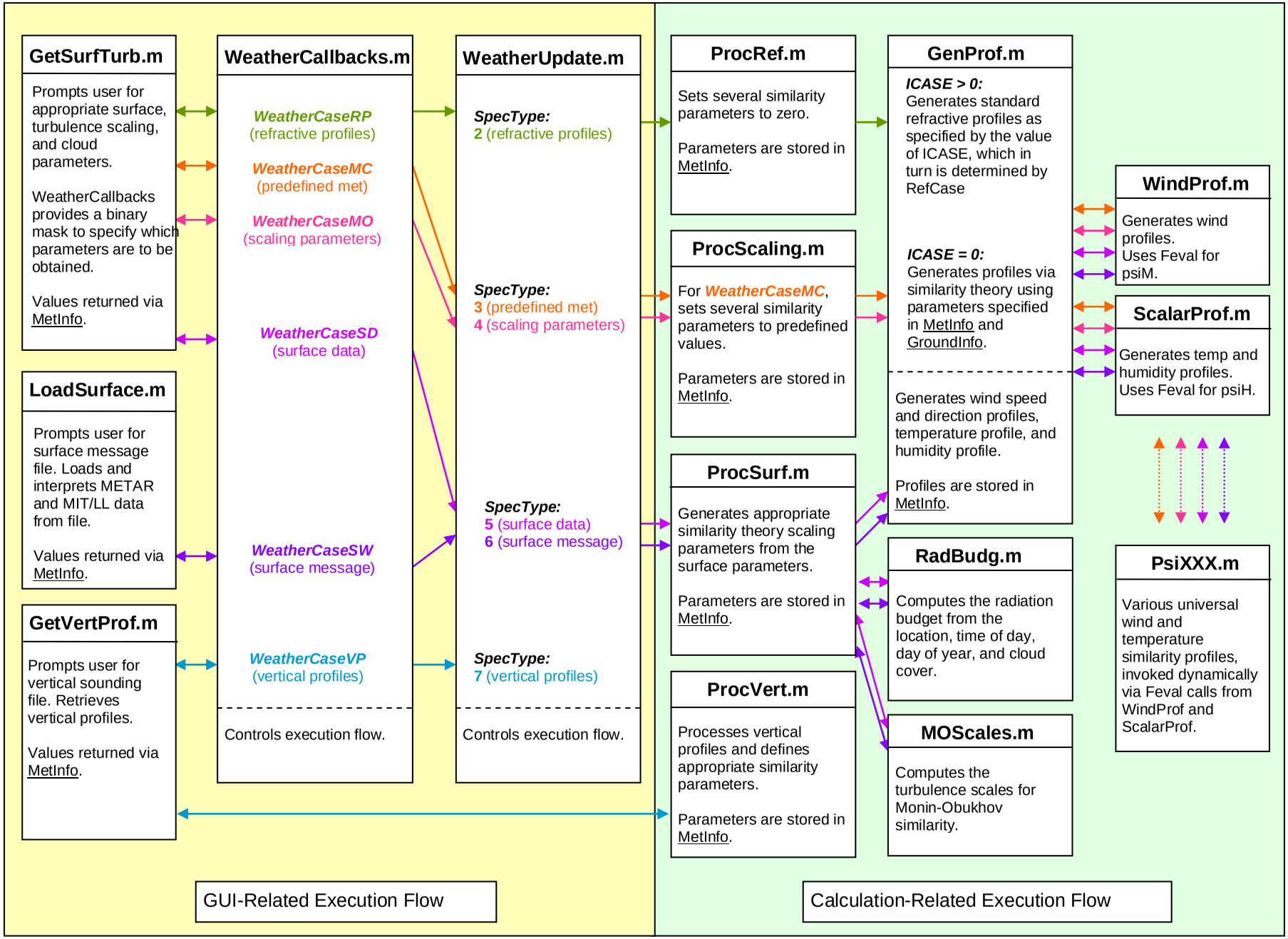

Figure C-1. SPEBE meteorological calculation execution flow diagram.

INTENTIONALLY LEFT BLANK.

Appendix D. CGroundInfo Supplemental Information

Table D-1 includes the GroundInfo fields encapsulated by **CGroundInfo**.

Table D-1. GroundInfo fields.

Field Name	Description	Type or Specified Values
Type	Name of predefined case	string
flowres	flow resistivity (cgs), for impedance	> 0
por	porosity, for impedance	0 <= por <= 1
sB	shape factor, for impedance	> 0
depth	(m), for impedance	
disp_hgt	displacement height (m), for similarity profile	> 0
albedo	for radiation characteristics	
emissivity	for radiation characteristics	
Bowen	for radiation characteristics	

INTENTIONALLY LEFT BLANK.

Appendix E. CSeismicInfo Supplemental Information

Table E-1 includes the SeismicInfo fields encapsulated by **CSeismicInfo**.

Table E-1. SeismicInfo fields.

Field Name	Description	Type or Specified Values
Type	Brief description	String
zint	Heights (m) of vertical profiles	Array of double
vp_prof	P-wave velocity (m/s) profile	Array of double
vs_prof	S-wave velocity (m/s) profile	Array of double
Qp_prof	P-wave attenuation profile	Array of double
Qs_prof	S-wave attenuation profile	Array of double
rho_prof	density profile	Array of double

INTENTIONALLY LEFT BLANK.

Appendix F. CNoizInfo Supplemental Information

Tables F-1 and F-2 include the NoizInfo Fields and the NoizInfo.Seismic and NoizInfo.Acoustic Fields, respectively, encapsulated by **CNoizInfo**.

Table F-1. NoizInfo fields.

Field Name	Description	Type or Specified Values
Acoustic	Acoustic background noise spectral description	Structure (see below)
IncWind	Include/don't include acoustic wind noise effects	0 – no wind; 1 - wind
Seismic	Seismic background noise spectral description	Structure (see below)

Table F-2. NoizInfo.seismic and NoizInfo.acoustic fields.

Field Name	Description	Type or Specified Values
Type	Identify acoustic or seismic	String: Acoustic Noise or Seismic Noise
SubType	Brief description	String
LowFreq	Array containing lower frequency of each band	Double array
HighFreq	Array containing upper frequency of each band	Double array
SpecSlope	Array containing spectral slope of each band	Double array
loudness	Array containing loudness of each band	Double array
OctaveBand	Use/don't use octave bands	0 – no octave bands; 1 – octave bands

INTENTIONALLY LEFT BLANK.

Appendix G. CTerrainInfo Supplemental Information

Figure G-1 includes the TerrainInfo fields encapsulated by **CTerrainInfo**.

Figure G-1. TerrainInfo fields.

Field Name	Description	Type or Specified Values
SpecType	Method used to generate/load terrain	2 – flat 3 – Gaussian (random) 4 – fractal (random) 5 – Digital Terrain Elevation Data (DTED) 6 – Digital Elevation Model (DEM) 7 – Georeferenced Tagged Image File Format (GeoTIFF) 8 - Compact Terrain Database (CTDB)
rmshgt	RMS height (meters), for Gaussian or fractal terrain	double
slope	Slope, for Gaussian or fractal terrain	double
meanhgt	mean height (meters), for Gaussian or fractal terrain	double

NOTE: RMS = root mean square

INTENTIONALLY LEFT BLANK.

Appendix I. CSourceInfo Supplemental Information

Tables I-1 and I-2 include the SourceInfo fields and the SourceInfo.AcsSpec and SourceInfo.SeisSpec fields encapsulated by **CSourceInfo**.

Table I-1. SourceInfo fields.

Field Name	Description	Type or Specified Values
Type	Main category (fixed wing, tracked heavy, etc)	string
SubType	Specific source name	string
xcoord	X coordinate of source in meters relative to origin	double
ycoord	Y coordinate of source in meters relative to origin	double
speed	Speed of source in meters/second	double
direc	Direction of source in degrees: 0 =east, 90 = north	double
rmsspeed	RMS speed in meters/second	double
zcoord	Height of source above ground, in meters	double
AcsSpec	Acoustic spectrum	structure (see below)
SeisSpec	Seismic spectrum	structure (see below)

Table I-2. SourceInfo.AcsSpec and SourceInfo.SeisSpec fields.

Field Name	Description	Type or Specified Values
LowFreq	Array containing lower frequency of each band	double array
HighFreq	Array containing upper frequency of each band	double array
SpecSlope	Array containing spectral slope of each band	double array
loudness	Array containing loudness of each band	double array
OctaveBand	Use/don't use octave bands	0 – no octave bands; 1 – octave bands
rad_pat	Radiation pattern	string: "monopole", "simple jet"

INTENTIONALLY LEFT BLANK.

Appendix J. CRcvrInfo Supplemental Information

Tables J-1 and J-2 include the RcvrInfo fields and the RcvrInfo.SensSpec and RcvrInfo.NoizSpec fields encapsulated by **CRcvrInfo**.

Table J-1. RcvrInfo fields.

Field Name	Description	Type or Specified Values
Type	Main category (linear array, polygonal array, etc)	string
SubType	Specific sensor name	string
xcoord	X coordinate of source in meters relative to origin	double
ycoord	Y coordinate of source in meters relative to origin	double
zcoord	Height of source above ground, in meters	double
spacing	Distance between elements, in meters	double
orient	Orientation of array, in degrees	double
NumSens	Number or elements in array	integer
WindScrDiam	Wind screen diameter	double
PD_algorithm	Detection probability algorithm, e.g. Neyman-Pearson	string
DF_algorithm	Direction finding algorithm e.g. Cramer-Rao lower bound	string
PD_param	Parameters associated with PD algorithm	array
DF_param	Parameters associated with Direction-Finding (DF) algorithm	array
PD_function	Matlab function to implement PD algorithm	string
DF_function	Matlab function to implement DF algorithm	string
SensPos	Coordinates of sensor elements relative to centroid	array
SensSpec	Sensor frequency response spectrum	Structure (see below)
NoizSpec	Sensor noise spectrum	Structure (see below)

Table J-2. RcvrInfo.SensSpec and RcvrInfo.NoizSpec fields.

Field Name	Description	Type or Specified Values
LowFreq	Array containing lower frequency of each band	double array
HighFreq	Array containing upper frequency of each band	double array
SpecSlope	Array containing spectral slope of each band	double array
loudness	Array containing loudness of each band	double array
OctaveBand	Use/don’t use octave bands	0 – no octave bands; 1 – octave bands
rad_pat	Radiation pattern	string: “monopole”, “simple jet”

INTENTIONALLY LEFT BLANK.

Appendix K. PerformanceCalculator Supplemental Information

Table K-1 includes the SPEBE parameters used by **PerformanceCalculator**.

Table K-1. SPEBE parameters used by **PerformanceCalculator.**

CalcCall calculations (DispType argument)	
Value	**Calculation**
'TL'	Transmission loss
'sat'	saturation
'coh'	coherence
'PD'	Probability of detection
'Pfa'	Probability of false alarm
'DF'	Direction-finding accuracy
'LFA'	Location-finding accuracy
'VFA'	Velocity-finding accuracy
'SNR'	Signal-to-noise ratio
'SAVE'	Generate and save propagation table

INTENTIONALLY LEFT BLANK.

Acronyms

2-D	two-dimensional
API	Application Programming Interface
CTDB	Compact Terrain Database
DEM	Digital Elevation Model
DF	Direction-Finding
DTED	Digital Terrain Elevation Data
GeoTIFF	Georeferenced Tagged Image File Format
GMT	Greenwich Mean Time
GUI	Graphical User Interface
MFC	Microsoft Foundation Class
PD	Probability of Detection
RMS	root mean square
SPEBE	Sensor Performance Evaluator for Battlefield Environments
UTC	Universal Time Coordinated

Distribution List

No. of Copies	Organization
1 elec	ADMNSTR DEFNS TECHL INFO CTR DTIC OCP (ELECTRONIC COPY) 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
5 CDs	US ARMY RSRCH LAB IMNE ALC IMS MAIL & RECORDS MGMT AMSRD ARL D J M MILLER AMSRD ARL CI OK TL TECHL LIB AMSRD ARL CI OK T TECHL PUB (2 copies) 2800 POWDER MILL ROAD ADELPHI MD 20783-1197
1 CD	US ARMY RESEARCH LAB AMSRD CI OK TP TECHL LIB ATTN T LANDFRIED APG MD 21005
1 CD	US ARMY RSRCH LAB DR D MARLIN AMSRD ARL CI EM WSMR NM 88002-5501
1 CD	US ARMY RSRCH LAB DR J NOBLE AMSRD ARL CI ES ADELPHI MD 20783-1197
1 CD	US ARMY RSRCH LAB DR S COLLIER AMSRD ARL CI ES ADELPHI MD 20783-1197
1 CD	US ARMY RSRCH LAB L PARKER AMSRD ARL CI EM ADELPHI MD 20783-1197
1 CD	US ARMY RSRCH LAB N SROUR AMSRD ARL SE SE ADELPHI MD 20783-1197
1 CD	ERDC CRREL SIGNATURE PHYSICS BRANCH ATTN DR K WILSON 72 LYME RD HANOVER NH 03755-1290
1 CD	US ARMY ARDEC AMSTA AR FSF R ATTN J CHANG BLDG 95N PICATINNY ARSENAL NJ 07806-5000
1 CD	RDECOM/ARDEC AMSRD AAR AEP A ATTN J HEBERLEY BLDG 407 PICATINNY ARSENAL NJ 07806-5000
15	Total: 1 electronic copy and 14 CDs